

# 1 Introduction

## 1.1 Overview

This is where we start, by looking at the human visual system to investigate what is meant by vision, how a computer can be made to sense pictorial data and how we can process an image. The overview of this Chapter is shown in Table 1.1; you will find a similar overview at the start of each chapter. References/citations are collected at the end of each Chapter.

Main topic	Sub topics	Main points
Human vision system	How the eye works, how visual information is processed and how it can fail.	<i>Sight, vision</i> , lens, retina, image, colour, monochrome, processing, brain, visual illusions.
Computer vision systems	How electronic images are formed, how video is fed into a computer and how we can process the information using a computer.	<i>Picture elements, pixels, video</i> standard, <i>camera</i> technologies, pixel technology, performance effects, specialist cameras, video conversion.
Processing Images	How we can process images using the Python computer language and mathematical packages; introduction to Python and to Matlab.	<i>Programming</i> and processing images, visualisation of results, availability, use.
Literature	Other textbooks and other places to find information on image processing, computer vision and feature extraction.	<i>Journals, textbooks</i> , websites and this book's website.

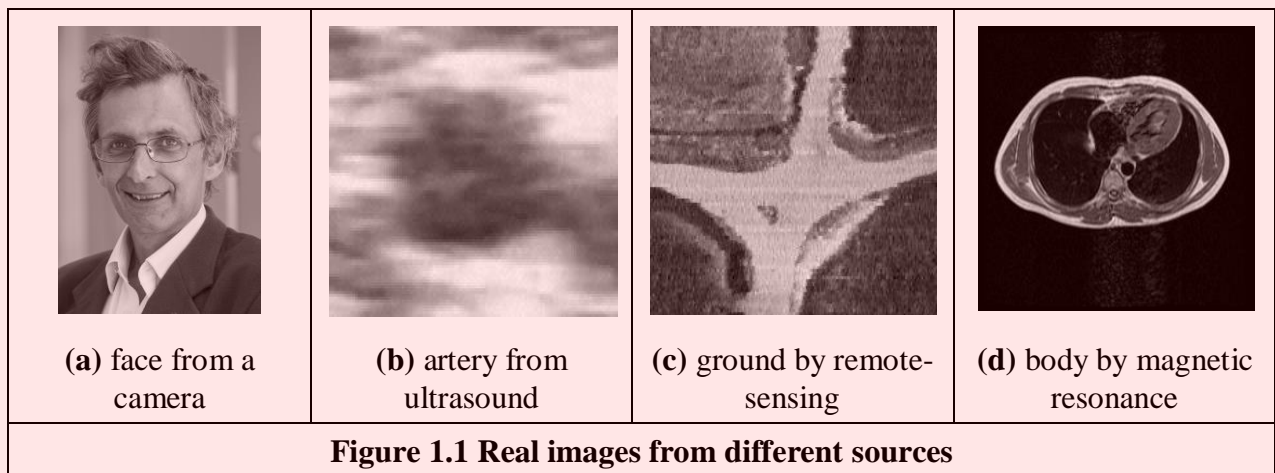
**Table 1.1 Overview of Chapter 1**

## 1.2 Human and Computer Vision

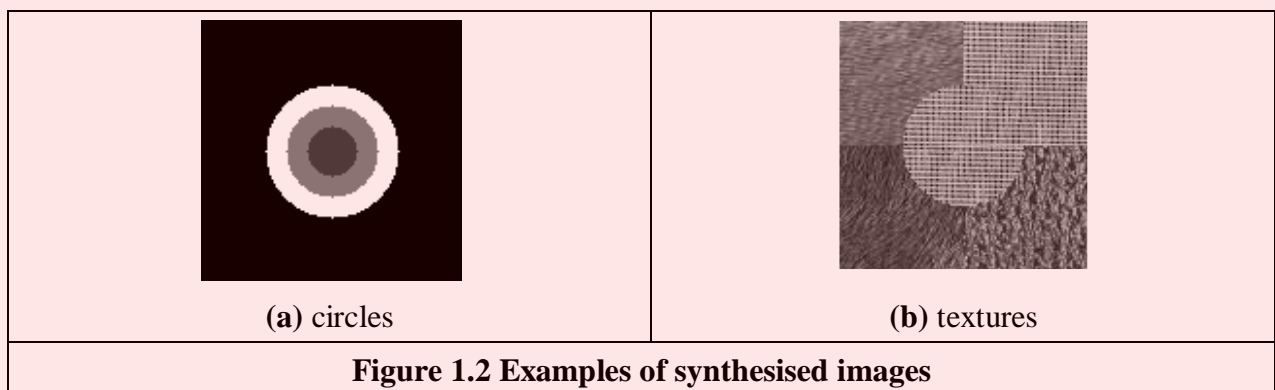
A computer vision system processes images acquired from an electronic camera, which is like the human vision system where the brain processes images derived from the eyes. Computer vision is a rich and rewarding topic for study and research for electronic engineers, computer scientists and many others. Now that cameras are cheap and widely available and computer power and memory are vast, computer vision is found in many places. There are now many vision systems in routine industrial use: cameras inspect mechanical parts to check size, food is inspected for quality, and images used in astronomy benefit from computer vision techniques. Forensic studies and biometrics (ways to recognise people) using computer vision include automatic face recognition and recognising people by the 'texture' of their irises. These studies are paralleled by biologists and psychologists who continue to study how our human vision system works, and how we see and recognise objects (and people).

A selection of (computer) images is given in Figure 1.1, these images comprise a set of points or *picture elements* (usually concatenated to *pixels*) stored as an array of numbers in a computer. To recognise faces, based on an image such as Figure 1.1(a), we need to be able to analyse constituent shapes, such as the shape of the nose, the eyes, and the eyebrows, to make some measurements to describe and then recognise a face. Figure 1.1(b) is an ultrasound image of the

carotid artery (which is near the side of the neck and supplies blood to the brain and the face), taken as a cross section through it. The top region of the image is near the skin; the bottom is inside the neck. The image arises from combinations of the reflections of the ultrasound radiation by tissue. This image comes from a study aimed to produce three-dimensional models of arteries, to aid vascular surgery. Note that the image is very noisy, and this obscures the shape of the (elliptical) artery. Remotely sensed images are often analysed by their texture content. The perceived texture is different between the road junction and the different types of foliage seen in Figure 1.1(c). Finally, Figure 1.1(d) is a Magnetic Resonance Image (MRI) of a cross-section near the middle of a human body. The chest is at the top of the image, and the lungs and blood vessels are the dark areas, the internal organs and the fat appears grey. MRI images are in routine medical use nowadays, owing to their ability to provide high quality images.



There are many different image sources. In medical studies, MRI is good for imaging soft tissue, but does not reveal the bone structure (the spine cannot be seen in Figure 1.1(d)); this can be achieved by using Computerised Tomography (CT) which is better at imaging bone, as opposed to soft tissue. Remotely sensed images can be derived from infrared (thermal) sensors or Synthetic-Aperture Radar, rather than by cameras, as in Figure 1.1(c). Spatial information can be provided by two-dimensional arrays of sensors, including sonar arrays. There are perhaps more varieties of sources of spatial data in medical studies than in any other area. But computer vision techniques are used to analyse any form of data, not just the images from cameras.



Synthesised images are good for evaluating techniques and finding out how they work, and some of the bounds on performance. Two synthetic images are shown in Figure 1.2. Figure 1.2(a) is an image of circles that were specified mathematically. The image is an ideal case: the circles are perfectly defined and the brightness levels have been specified to be constant. This type of synthetic image is good for evaluating techniques which find the borders of the shape (its edges), the shape itself and even for making a description of the shape. Figure 1.2(b) is a synthetic image made up of

sections of real image data. The borders between the regions of image data are exact, again specified by a program. The image data comes from a well-known texture database, the Brodatz album of textures. This was scanned and stored as a computer image. This image can be used to analyse how well computer vision algorithms can identify regions of differing texture.

This Chapter will show you how basic computer vision systems work, in the context of the human vision system. It covers the main elements of human vision showing you how your eyes work (and how they can be deceived!). For computer vision, this Chapter covers the hardware and the software used for image analysis, giving an introduction to Python and Matlab, the software and mathematical packages, respectively, used throughout this text to implement computer vision algorithms. Finally, a selection of pointers to other material is provided, especially those for more detail on the topics covered in this Chapter.

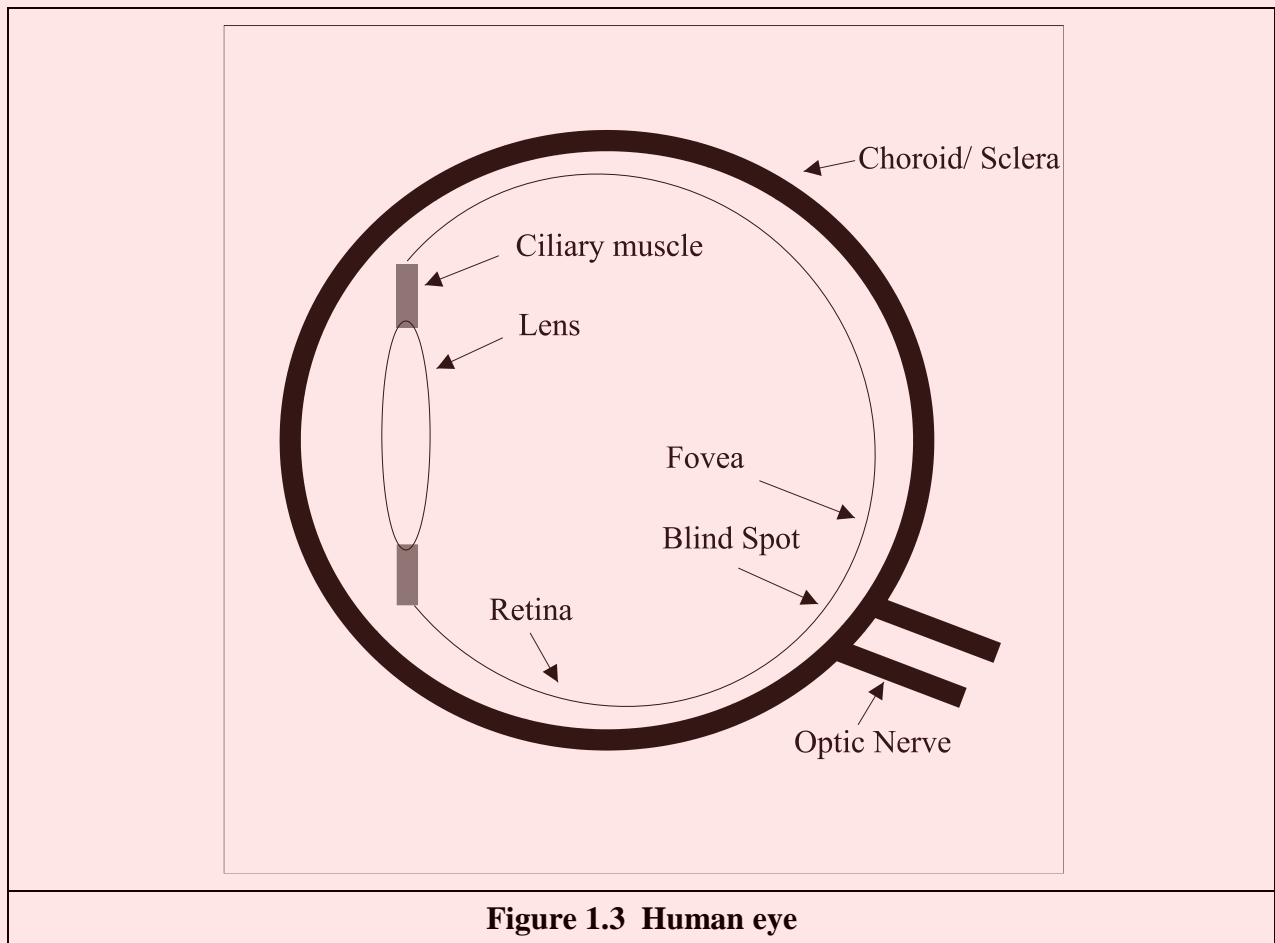
### 1.3 The Human Vision System

*Human vision* is a sophisticated system that senses and acts on visual stimuli. It has evolved for millions of years, primarily for defence or survival. Intuitively, computer and human vision appear to have the same function. The purpose of both systems is to interpret spatial data, data that is indexed by more than one dimension. Even though computer and human vision are functionally similar, you cannot expect a computer vision system to exactly replicate the function of the human eye. This is partly because we do not understand fully how the vision system of the eye and brain works, as we shall see in this section. Accordingly, we cannot design a system to exactly replicate its function. In fact, some of the properties of the human eye are useful when developing computer vision techniques, whereas others are actually undesirable in a computer vision system. But we shall see computer vision techniques which can to some extent replicate - and in some cases even improve upon - the human vision system.

You might ponder this, so put one of the fingers from each of your hands in front of your face and try to estimate the distance between them. This is difficult, and we are sure you would agree that your measurement would not be very accurate. Now put your fingers very close together. You can still tell that they are apart even when the distance between them is tiny. So human vision can distinguish relative distance well, but is poor for absolute distance. Computer vision is the other way around: it is good for estimating absolute difference, but with relatively poor resolution for relative difference. The number of pixels in the image imposes the accuracy of the computer vision system, but that does not come until the next Chapter. Let us start at the beginning, by seeing how the human vision system works.

In human vision, the sensing element is the eye from which images are transmitted via the optic nerve to the brain, for further processing. The optic nerve has insufficient bandwidth to carry all the information sensed by the eye. Accordingly, there must be some pre-processing before the image is transmitted down the optic nerve. The human vision system can be modelled in three parts:

- 1) the eye – this is a physical model since much of its function can be determined by pathology;
- 2) a processing system – this is an experimental model since the function can be modelled, but not determined precisely; and
- 3) analysis by the brain – this is a psychological model since we cannot access or model such processing directly, but only determine behaviour by experiment and inference.



### 1.3.1 The Eye

The function of the eye is to form an image; a cross-section of the eye is illustrated in Figure 1.3. Vision requires an ability to selectively focus on objects of interest. This is achieved by the *ciliary muscles* that hold the *lens*. In old age, it is these muscles which become slack and the eye loses its ability to focus at short distance. The *iris*, or pupil, is like an aperture on a camera and controls the amount of light entering the eye. It is a delicate system and needs protection, this is provided by the cornea (sclera). This is outside the *choroid* which has blood vessels that supply nutrition and is opaque to cut down the amount of light. The *retina* is on the inside of the eye, which is where light falls to form an image. By this system, muscles rotate the eye, and shape the lens, to form an image on the *fovea* (focal point) where the majority of sensors are situated. The *blind spot* is where the optic nerve starts, there are no sensors there.

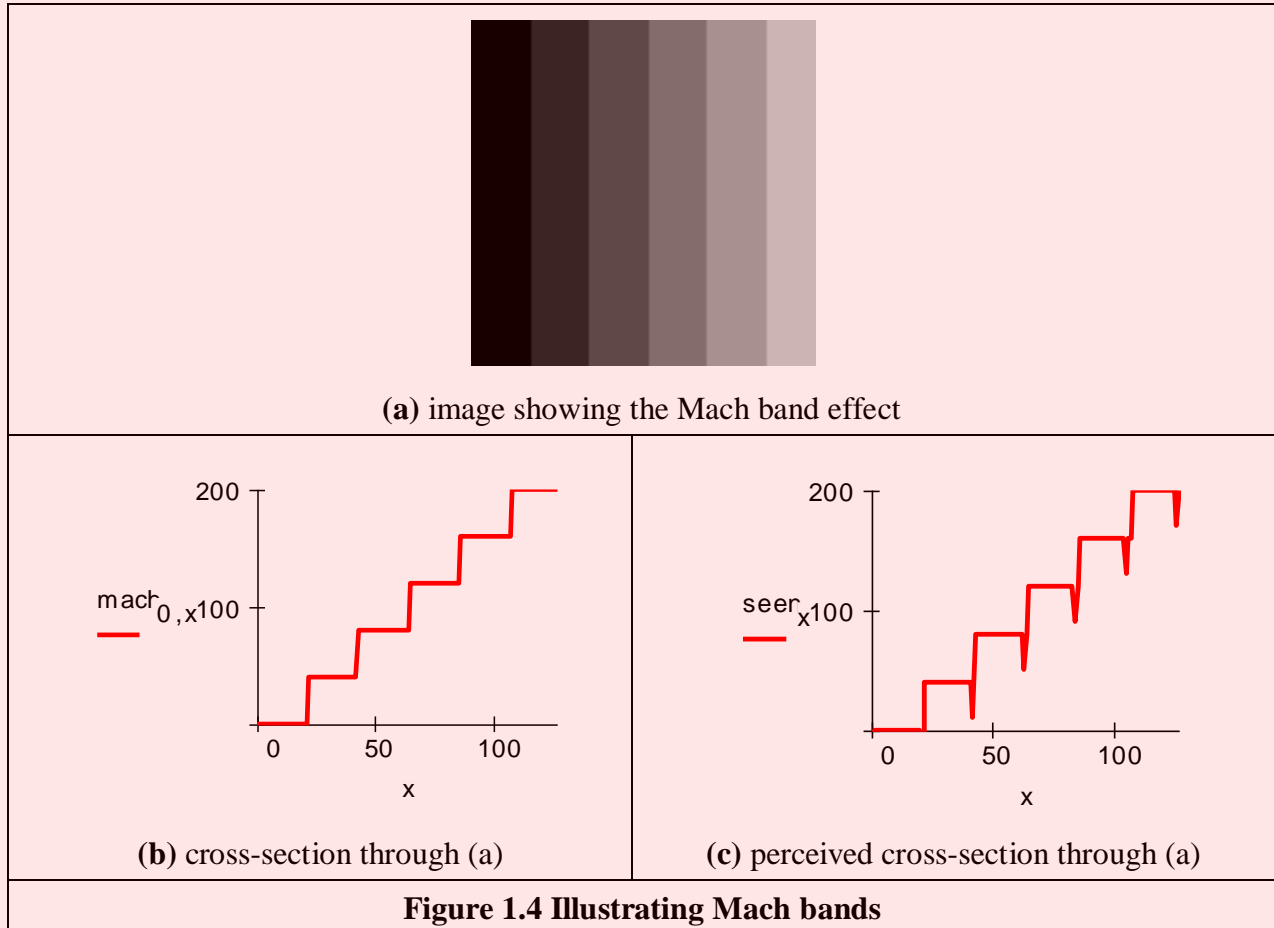
Focusing involves shaping the lens, rather than positioning it as in a camera. The lens is shaped to refract close images greatly, and distant objects little, essentially by ‘stretching’ it. The distance of the focal centre of the lens varies from approximately 14 mm to around 17 mm depending on the lens shape. This implies that a world scene is translated into an area of about 2 mm<sup>2</sup>. Good vision has high *acuity* (sharpness), which implies that there must be very many sensors in the area where the image is formed.

There are actually nearly 100 million sensors dispersed around the retina. Light falls on these sensors to stimulate photochemical transmissions, which results in nerve impulses that are collected to form the signal transmitted by the eye. There are two types of sensor: firstly the *rods* – these are used for black and white (*scotopic*) vision; and secondly the *cones*- these are used for colour (*photopic*) vision. There are approximately 10 million cones and nearly all are found within 5° of the fovea. The remaining 100 million rods are distributed around the retina, with the majority

between  $20^\circ$  and  $5^\circ$  of the fovea. Acuity is actually expressed in terms of spatial resolution (sharpness) and brightness/ colour resolution and is greatest within  $1^\circ$  of the fovea.

There is only one type of rod, but there are three types of cones. These types are:

1. S – short wavelength: these sense light towards the blue end of the visual spectrum;
2. M – medium wavelength: these sense light around green; and
3. L – long wavelength: these sense light towards the red region of the spectrum.

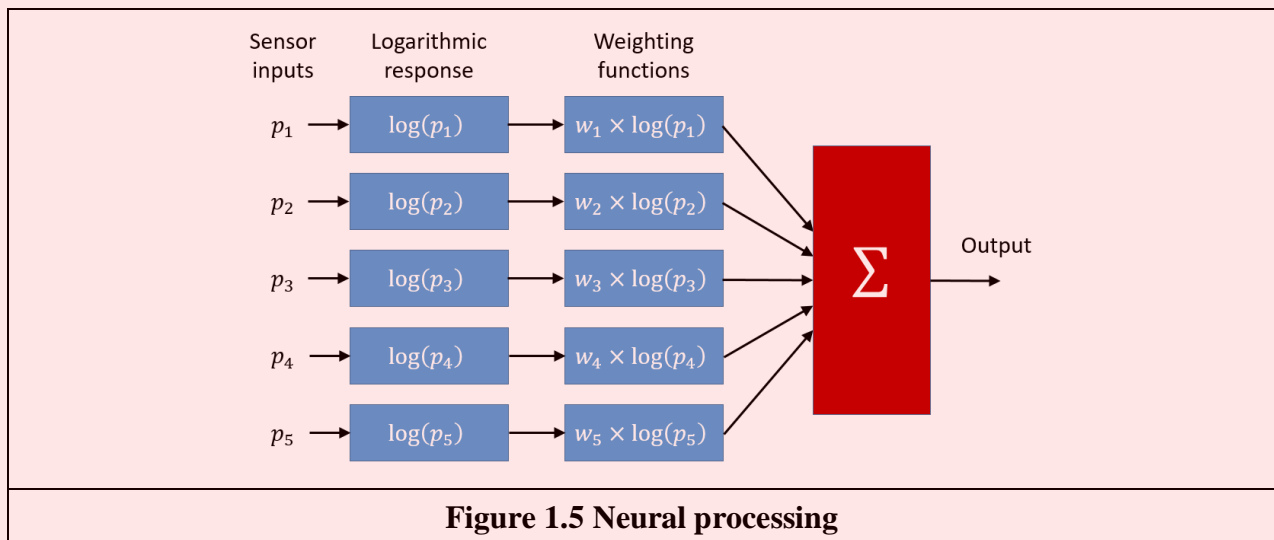


The total response of the *cones* arises from summing the response of these three types of cone, this gives a response covering the whole of the visual spectrum. The rods are sensitive to light within the entire visual spectrum, giving the monochrome capability of scotopic vision. When the light level is low, images are formed away from the fovea to use the superior sensitivity of the rods, but without the colour vision of the cones. Note that there are actually very few of the blueish cones, and there are many more of the others. But we can still see a lot of blue (especially given ubiquitous denim!). So, somehow, the human vision system compensates for the lack of blue sensors, to enable us to perceive it. The world would be a funny place with red water! The vision response is actually logarithmic and depends on brightness adaption from dark conditions where the image is formed on the rods, to brighter conditions where images are formed on the cones. More on colour sensing is to be found in Chapter 11.

One inherent property of the eye, known as *Mach bands*, affects the way we perceive images. These are illustrated in Figure 1.4 and are the bands that appear to be where two stripes of constant shade join. By assigning values to the image brightness levels, the cross-section of plotted brightness is shown in Figure 1.4(a). This shows that the picture is formed from stripes of constant brightness. Human vision perceives an image for which the cross-section is as plotted in Figure 1.4(c). These Mach bands do not really exist, but are introduced by your eye. The bands arise from

overshoot in the eyes' response at boundaries of regions of different intensity (this aids us to differentiate between objects in our field of view). The real cross-section is illustrated in Figure 1.4(b). Note also that a human eye can distinguish only relatively few grey levels. It actually has a capability to discriminate between 32 levels (equivalent to five bits) whereas the image of Figure 1.4(a) could have many more brightness levels. This is why your perception finds it more difficult to discriminate between the low intensity bands on the left of Figure 1.4(a). (Note that Mach bands cannot be seen in the earlier image of circles, Figure 1.2(a), due to the arrangement of grey levels.) This is the limit of our studies of the first level of human vision; for those who are interested; [Cornsweet70] provides many more details concerning visual perception.

So we have already identified two properties associated with the eye that it would be difficult to include, and would often be unwanted, in a computer vision system: Mach bands and sensitivity to unsensed phenomena. These properties are integral to human vision. At present, human vision is far more sophisticated than we can hope to achieve with a computer vision system. Infrared guided-missile vision systems can actually have difficulty in distinguishing between a bird at 100 m and a plane at 10 km. Poor birds! (Lucky plane?) Human vision can handle this with ease.



**Figure 1.5 Neural processing**

### 1.3.2 The Neural System

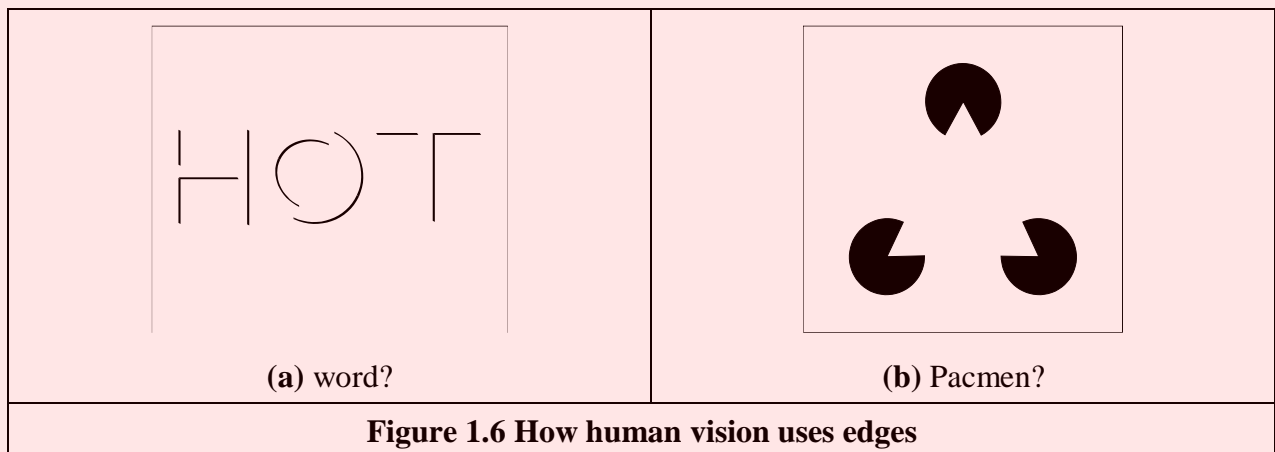
*Neural signals* provided by the eye are essentially the transformed response of the wavelength dependent receptors, the cones and the rods. One model is to combine these transformed signals by addition, as illustrated in Figure 1.5. The response is transformed by a logarithmic function, mirroring the known response of the eye. This is then multiplied by a weighting factor that controls the contribution of a particular sensor. This can be arranged to allow combination of responses from a particular region. The weighting factors can be chosen to afford particular filtering properties. For example, in *lateral inhibition*, the weights for the centre sensors are much greater than the weights for those at the extreme. This allows the response of the centre sensors to dominate the combined response given by addition. If the weights in one half are chosen to be negative, whilst those in the other half are positive, then the output will show detection of contrast (change in brightness), given by the differencing action of the weighting functions.

The signals from the cones can be combined in a manner that reflects *chrominance* (colour) and *luminance* (brightness). This can be achieved by subtraction of logarithmic functions, which is then equivalent to taking the logarithm of their ratio. This allows measures of chrominance to be obtained. In this manner, the signals derived from the sensors are combined prior to transmission through the optic nerve. This is an experimental model, since there are many ways possible to combine the different signals together.

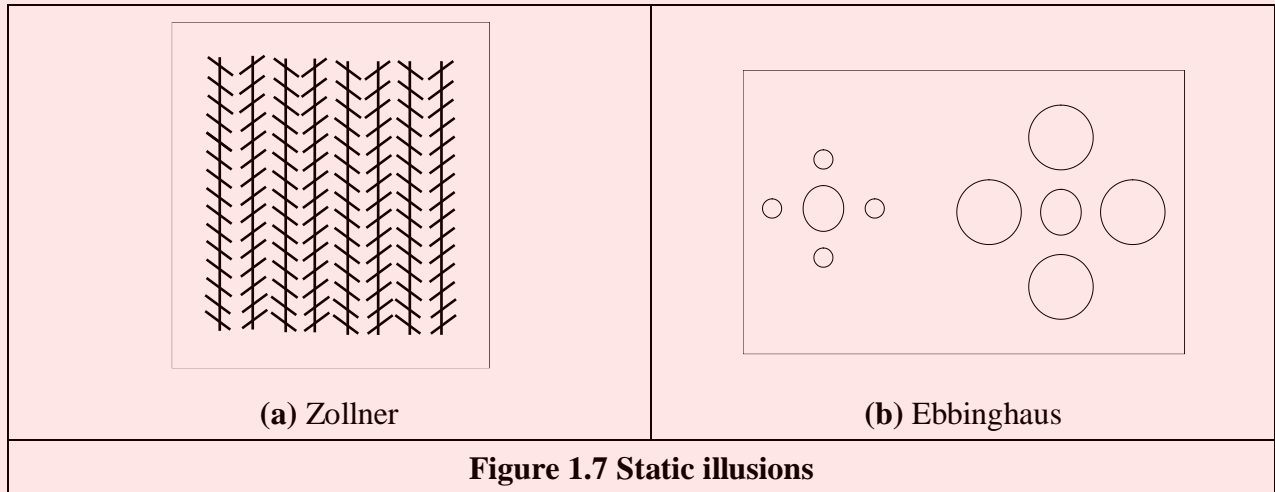
Visual information is then sent back to arrive at the *Lateral Geniculate Nucleus* (LGN) which is in the thalamus and is the primary processor of visual information. This is a layered structure containing different types of cells, with differing functions. The axons from the LGN pass information on to the visual cortex. The function of the LGN is largely unknown, though it has been shown to play a part in coding the signals that are transmitted. It is also considered to help the visual system focus its attention, such as on sources of sound. For further information on retinal neural networks, see [Ratliff65]; an alternative study of neural processing can be found in [Overington92].

### 1.3.3 Processing

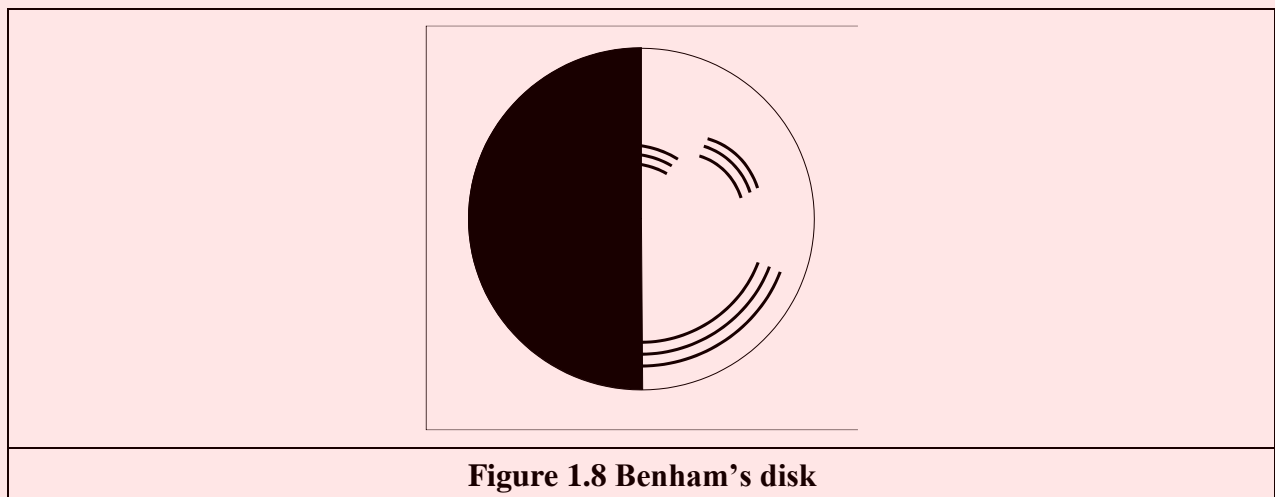
The neural signals are then transmitted to two areas of the brain for further processing. These areas are the *associative cortex*, where links between objects are made, and the *occipital cortex*, where patterns are processed. It is naturally difficult to determine precisely what happens in this region of the brain. To date, there have been no volunteers for detailed study of their brain's function (though progress with new imaging modalities such as Positive Emission Tomography or Electrical Impedance Tomography will doubtless help). For this reason, there are only psychological models to suggest how this region of the brain operates.



It is well known that one function of the human vision system is to use edges, or boundaries, of objects. We can easily read the word in Figure 1.6(a), this is achieved by filling in the missing boundaries in the knowledge that the pattern most likely represents a printed word. But we can infer more about this image; there is a suggestion of illumination, causing shadows to appear in unlit areas. If the light source is bright, then the image will be washed out, causing the disappearance of the boundaries which are interpolated by our eyes. So there is more than just physical response, there is also knowledge, including prior knowledge of solid geometry. This situation is illustrated in Figure 1.6(b) that could represent three 'pacmen' about to collide, or a white triangle placed on top of three black circles. Either situation is possible.



It is also possible to deceive human vision, primarily by imposing a scene that it has not been trained to handle. In the famous *Zollner illusion*, Figure 1.7(a), the bars appear to be slanted, whereas in reality they are vertical (check this by placing a pen between the lines): the small crossbars mislead your eye into perceiving the vertical bars as slanting. In the *Ebbinghaus illusion*, Figure 1.7(b), the inner circle appears to be larger when surrounded by small circles, than it is when surrounded by larger circles.



There are dynamic illusions too: you can always impress children with the “see my wobbly pencil” trick. Just hold the pencil loosely between your fingers then, to whoops of childish glee, when the pencil is shaken up and down, the solid pencil will appear to bend. *Benham's disk*, Figure 1.8, shows how hard it is to model vision accurately. If you make up a version of this disk into a spinner (push a matchstick through the centre) and spin it anti-clockwise, you do not see three dark rings, you will see three coloured ones. The outside one will appear to be red, the middle one a sort of green, and the inner one will appear deep blue. (This can depend greatly on lighting - and contrast between the black and white on the disk. If the colours are not clear, try it in a different place, with different lighting.) You can appear to explain this when you notice that the red colours are associated with the long lines, and the blue with short lines. But that is from physics, not psychology. Now spin the disk clockwise. The order of the colours reverses: red is associated with the short lines (inside), and blue with the long lines (outside). So the argument from physics is clearly incorrect, since red is now associated with short lines not long ones, revealing the need for psychological explanation of the eyes' function. This is not colour perception, see [Armstrong91] for an interesting (and interactive!) study of colour theory and perception.



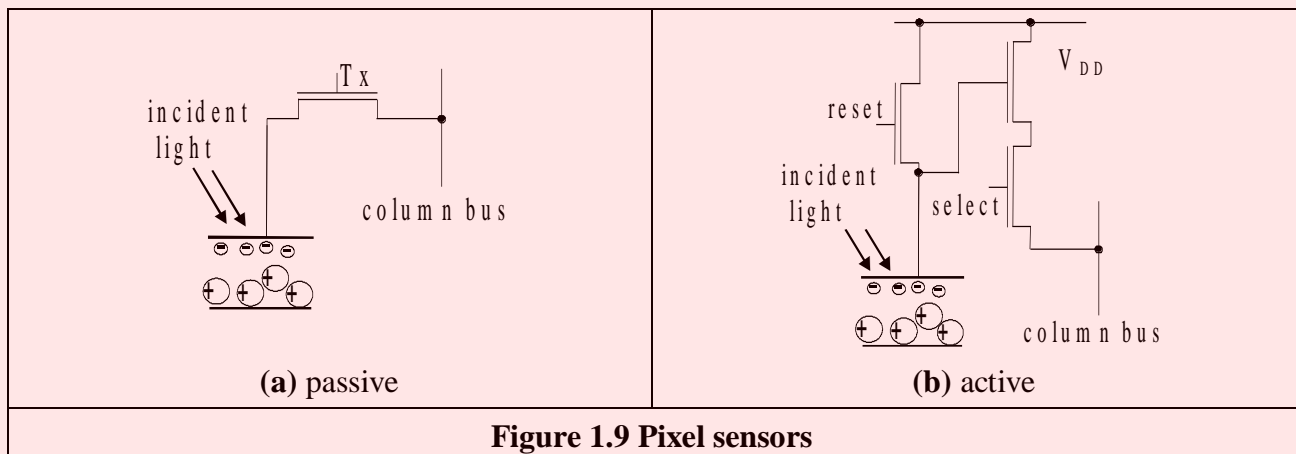
Naturally, there are many texts on human vision – one popular text on human visual perception (and its relationship with visual art) is by Livingstone [Livingstone14]; there is an online book: *The Joy of Vision* (<http://www.yorku.ca/eye/thejoy.htm>) – useful, despite its title! Marr’s seminal text [Marr82] is a computational investigation into human vision and visual perception, investigating it from a computer vision viewpoint. For further details on pattern processing in human vision, see [Bruce90]; for more illusions see [Rosenfeld82] and an excellent – and dynamic – collection at <https://michaelbach.de/ot>. Many of the properties of human vision are hard to include in a computer vision system, but let us now look at the basic components that are used to make computers see.

## 1.4 Computer Vision Systems

Given the progress in computer technology and domestic photography, computer vision hardware is now relatively inexpensive; a basic computer vision system requires a camera, a camera interface and a computer. These days, many personal computers offer the capability for a basic vision system, by including a camera and its interface within the system. There are specialised systems for computer vision, offering high performance in more than one aspect. These can be expensive, as any specialist system is.

### 1.4.1 Cameras

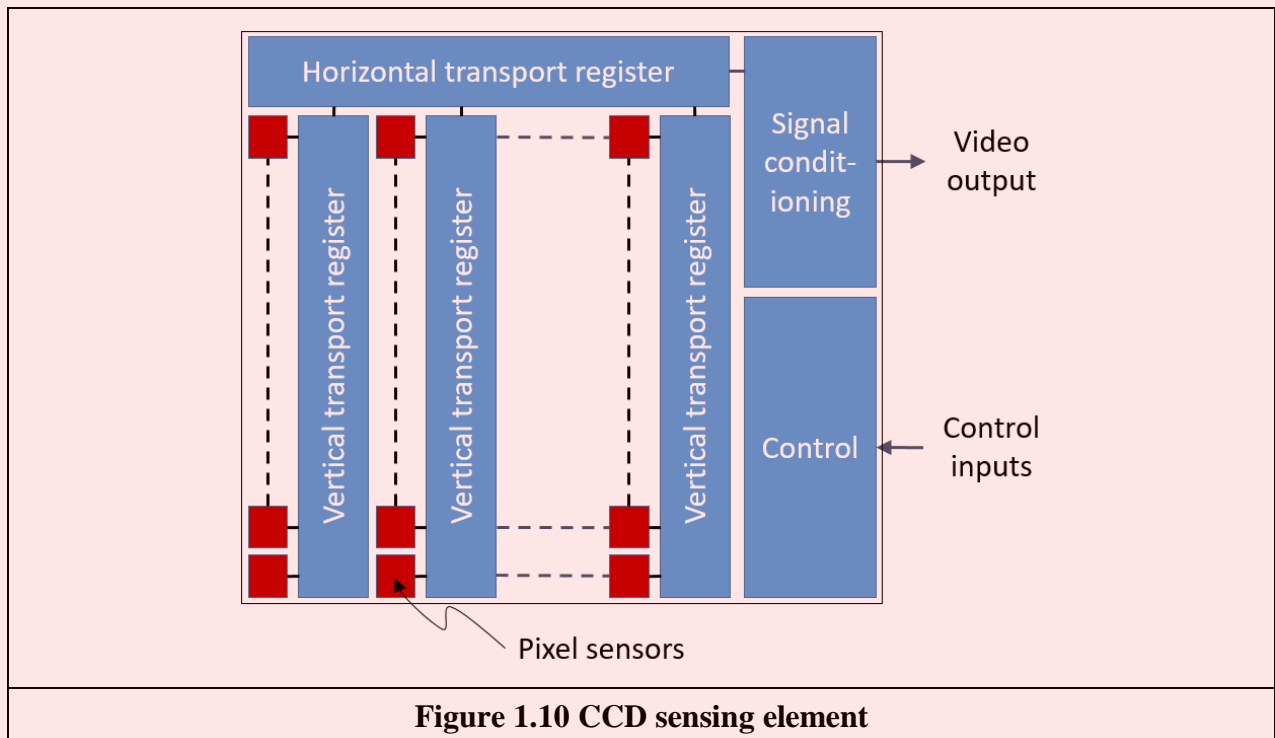
A *camera* is the basic sensing element. In simple terms, most cameras rely on the property of light to cause hole/electron pairs (the charge carriers in electronics) in a conducting material. When a potential is applied (to attract the charge carriers), this charge can be sensed as current. By Ohm’s law, the voltage across a resistance is proportional to the current through it, so the current can be turned in to a voltage by passing it through a resistor. The number of hole/electron pairs is proportional to the amount of incident light. Accordingly, greater charge (and hence greater voltage and current) is caused by an increase in brightness. In this manner cameras can provide as output, a voltage which is proportional to the brightness of the points imaged by the camera.



There are three main types of camera: *vidicons*, *charge coupled devices* (CCDs) and, later, *CMOS* cameras (Complementary Metal Oxide Silicon - now the dominant technology for logic circuit implementation). Vidicons are the old (analogue) technology, which though cheap (mainly by virtue of longevity in production) have largely been replaced by the newer CCD and CMOS digital technologies. The digital technologies now dominate much of the camera market because they are lightweight and cheap (with other advantages) and are therefore used in the domestic video market.

Vidicons operate in a manner akin to an old television in reverse. The image is formed on a screen, and then sensed by an electron beam that is scanned across the screen. This produces an output which is continuous, the output voltage is proportional to the brightness of points in the scanned line, and is a continuous signal, a voltage which varies continuously with time. On the other hand, CCDs and CMOS cameras use an array of sensors; these are regions where charge is collected which is proportional to the light incident on that region. This is then available in discrete, or sampled, form as opposed to the continuous sensing of a vidicon. This is similar to human vision with its array of cones and rods, but digital cameras use a rectangular regularly spaced lattice whereas human vision uses a hexagonal lattice with irregular spacing.

Two main types of semiconductor pixel sensors are illustrated in Figure 1.9. In the *passive sensor*, the charge generated by incident light is presented to a bus through a pass transistor. When the signal Tx is activated, the pass transistor is enabled and the sensor provides a capacitance to the bus, one that is proportional to the incident light. An *active pixel* includes an amplifier circuit that can compensate for the limited fill factor of the photodiode. The select signal again controls presentation of the sensor's information to the bus. A further reset signal allows the charge site to be cleared when the image is re-scanned.



**Figure 1.10 CCD sensing element**

The basis of a CCD sensor is illustrated in Figure 1.10. The number of charge sites gives the resolution of the CCD sensor; the contents of the charge sites (or buckets) need to be converted to an output (voltage) signal. In simple terms, the contents of the buckets are emptied into vertical transport registers which are shift registers moving information towards the horizontal transport registers. This is the column bus supplied by the pixel sensors. The horizontal transport registers empty the information row by row (point by point) into a signal conditioning unit which transforms the sensed charge into a voltage which is proportional to the charge in a bucket, and hence proportional to the brightness of the corresponding point in the scene imaged by the camera. CMOS cameras are like a form of memory: the charge incident on a particular site in a two-dimensional lattice is proportional to the brightness at a point. The charge is then read like computer memory. (In fact, a computer memory RAM chip can act as a rudimentary form of camera when the circuit - the one buried in the chip - is exposed to light.)

There are many more varieties of vidicon (Chalnicon etc.) than there are of CCD technology (Charge Injection Device etc.), perhaps due to the greater age of basic vidicon technology. Vidicons are cheap but have a number of intrinsic performance problems. The scanning process essentially relies on ‘moving parts’. As such, the camera performance will change with time, as parts wear; this is known as *ageing*. Also, it is possible to *burn* an image into the scanned screen by using high incident light levels; vidicons can also suffer *lag* that is a delay in response to moving objects in a scene. On the other hand, the digital technologies are dependent on the physical arrangement of charge sites and as such do not suffer from ageing, but can suffer from irregularity in the charge sites’ (silicon) material. The underlying technology also makes CCD and CMOS cameras less sensitive to lag and burn, but the signals associated with the CCD transport registers can give rise to *readout effects*. CCDs actually only came to dominate camera technology when technological difficulty associated with *quantum efficiency* (the magnitude of response to incident light) for the shorter, blue, wavelengths were solved. One of the major problems in CCD cameras is *blooming* where bright (incident) light causes a bright spot to grow and disperse in the image (this used to happen in the analog technologies too). This happens much less in CMOS cameras because the charge sites can be much better defined and reading their data is equivalent to reading memory sites as opposed to shuffling charge between sites. Also, CMOS cameras have now overcome the problem of *fixed pattern noise* that plagued earlier MOS cameras. CMOS cameras are actually much more recent than CCDs. This begs a question as to which is best: CMOS or CCD? An early view was that CCD could provide higher quality images whereas CMOS is a cheaper technology and because it lends itself directly to intelligent cameras with on-board processing. The feature size of points (pixels) in a CCD sensor is limited to be about 4  $\mu\text{m}$  so that enough light is collected. In contrast, the feature size in CMOS technology is considerably smaller. It is then possible to integrate signal processing within the camera chip and thus it is perhaps possible that CMOS cameras will eventually replace CCD technologies for many applications. However, modern CCDs’ process technology is more mature, so the debate will doubtless continue!

Finally, there are specialist cameras, which include high-resolution devices (giving pictures with many points), low-light level cameras which can operate in very dark conditions and *infrared* cameras which sense heat to provide thermal images; *hyperspectral* cameras have more sensing bands. For more detail concerning modern camera practicalities and imaging systems see [Nakamura05] and more recently [Kuroda14]. For more detail on sensor development, particularly CMOS, [Fossum97] is still well worth a look. For more detail on images see [Phillips18] with a particular focus on quality (hey - there is even mosquito noise!).

A *light field* – or *plentoptic* - camera is one that can sense depth as well as brightness [Adelson05]. The light field is essentially a two dimensional set of spatial images, thus giving a 4-dimensional array of pixels. The light field can be captured in a number of ways, by moving cameras, or multiple cameras. The aim is to capture the plenoptic function that describes the light as a function of position, angle, wavelength and time [Wu17]. These days, commercially available cameras use lenses to derive the light field. These can be used to render an image into full depth of plane focus (imagine an image of an object taken at close distance where only the object is in focus combined with an image where the background is in focus to give an image where both the object and the background are in focus). A surveillance operation could focus on what is behind an object which would show in a normal camera image. This gives an alternative approach to 3D object analysis, by sensing the object in 3D. Wherever there are applications, industry will follow, and that has proved to be the case.

There are new *dynamic vision sensors* which sense motion [Lichtsteiner08, Son17] and are much closer to the starting grid than the light field cameras. Clearly, the resolution and speed continue to improve, and there are applications emerging that use these sensors. We shall find in Chapters 4 and 9 it is possible to estimate motion from sequences of images. These sensors are different, since

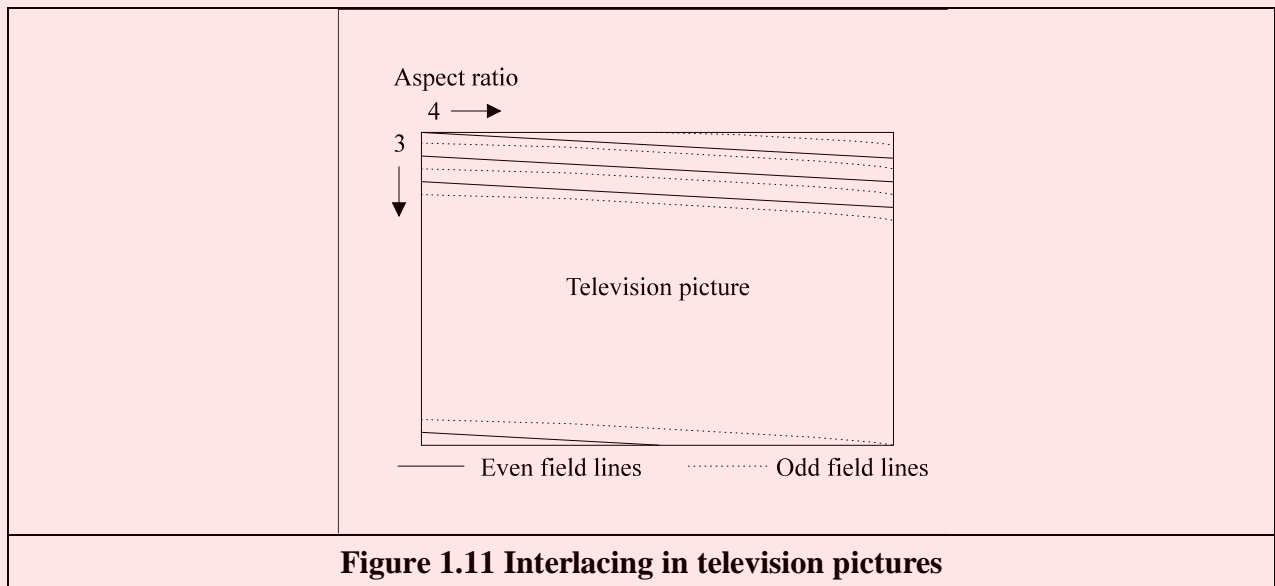
they specifically target motion. As the target application is security (much security video is dull stuff indeed, with little motion) allowing recording only of material of likely interest.

### 1.4.2 Computer Interfaces

Though digital cameras continue to advance there are still some legacies from the older analog systems to be found in the some digital systems. There is also some older technology in deployed systems. As such, we shall cover the main points of the two approaches. Essentially, the image sensor converts light into a signal which is expressed either as a continuous signal, or in sampled (digital) form. Some (older) systems expressed the camera signal as an analog continuous signal, according to a standard and this was converted at the computer (and still is in some cases, using a framegrabber). Modern digital systems convert the sensor information into digital information with on-chip circuitry and then provide the digital information according to a specified standard. The older systems, such as surveillance systems, supplied (or supply) video whereas the newer systems are digital. Video implies delivering the moving image as a sequence of *frames* of which one format is *Digital Video – DV*.

An analog continuous camera signal is transformed into digital (discrete) format using an Analogue to Digital (A/D) converter. *Flash converters* are usually used due to the high speed required for conversion (say 11 MHz that cannot be met by any other conversion technology). Usually, 8-bit A/D converters are used; at 6dB/ bit, this gives 48dB which just satisfies the CCIR stated *bandwidth* of approximately 45dB. The outputs of the A/D converter are then stored. Note that there are aspects of the sampling process which are of considerable interest in computer vision; these are covered in Chapter 2.

In digital camera systems this processing is usually performed on the camera chip, and the camera eventually supplies digital information, often in coded form. Currently, Thunderbolt is the hardware interface that dominates the high end of the market and USB is used at the lower end. There was a system called Firewire but it has now faded. Images are constructed from a set of lines, those lines scanned by a camera. In the older analog systems, in order to reduce requirements on transmission (and for viewing), the 625 lines (in the *PAL* system, *NTSC* is of lower resolution) were transmitted in two *interlaced fields*, each of 312.5 lines, as illustrated in Figure 1.11. These were the odd and the even fields. Modern televisions are *progressive scan*, which is like reading a book: the picture is constructed line by line. There is also an *aspect ratio* in picture transmission: pictures are arranged to be longer than they are high. These factors are chosen to make television images attractive to human vision. Nowadays, *digital video cameras* can provide digital output, in progressive scan delivering sequences of images that are readily processed. There are Gigabit Ethernet cameras which transmit high speed video and control information over Ethernet networks. Or there are *webcams*, or just *digital camera systems* that deliver images straight to the computer. Life just gets easier!



## 1.5 Processing Images

We shall be using software and packages to process computer images. There are many programming languages and we have chosen Python as perhaps the most popular language at the time of writing. Several *mathematical systems* allow you to transpose mathematics more easily from textbooks, and see how it works. Code functionality is not obscured by the use of data structures, though this can make the code appear cumbersome (to balance though, the range of datatypes is invariably small). A major advantage of processing packages is that they provide the low-level functionality and data visualisation schemes, allowing the user to concentrate on techniques alone. Accordingly, these systems afford an excellent route to understand, and appreciate, mathematical systems prior to development of application code, and to check the final code works correctly. The packages are however less suited to the development of application code. Chacun à son gout!

The code that supports this book has been written for educational purposes only. We encourage you to use it for that purpose, for that is one of the best ways of learning. It is written for clarity, aiming to support the text here, and is not optimised in any way. You might use the code elsewhere: you are welcome, naturally, though we admit no liability of any form. Our target here is education, and nothing else.

### 1.5.1 Processing

Most image processing and computer vision techniques are implemented in computer software. Often, only the simplest techniques migrate to hardware; though coding techniques to maximise efficiency in image transmission are of sufficient commercial interest that they have warranted extensive, and very sophisticated, hardware development. The systems include the *Joint Photographic Expert Group* (JPEG) and the *Moving Picture Expert Group* (MPEG) image coding formats. C, C++, Python and Java<sup>TM</sup> are by now the most popular languages for vision system implementation, because of strengths in integrating high- and low-level functions, and the availability of good compilers. As systems become more complex, C++, Python and Java become more attractive when encapsulation and polymorphism may be exploited. Many people use Python and JAVA as a development language partly due to platform independence, but also due to ease in implementation (though some claim that speed/ efficiency is better in C/C++). Python is currently

a language of choice, not for any specific reasons. Some are up and coming, so there are advocates for JULIA too. There are some textbooks that offer image processing systems implemented in these languages. Also, there are many commercial packages available, though these are often limited to basic techniques, and do not include the more sophisticated shape extraction techniques – and the underlying implementation can be hard to check. Some popular texts present working algorithms, such as [O’Gorman08], [Parker10] and [Solem12].

In terms of software packages, the most popular is OpenCV (Open Source Computer Vision) whose philosophy is to “aid commercial uses of computer vision in human-computer interface, robotics, monitoring, biometrics and security by providing a free and open infrastructure where the distributed efforts of the vision community can be consolidated and performance optimized”. This contains a wealth of technique and (optimised) implementation – there’s a Wikipedia entry and a discussion website supporting it. The system has now moved to OpenCV 4. That means there were versions 1, 2 and 3 and unfortunately the systems are not compatible. The first textbook describing its use [Bradski08] had/has excellent descriptions of how to use the code (and some great diagrams) but omitted much of the (mathematical) background and analysis so it largely describes usage rather than construction. There are more recent texts for OpenCV, but the reviews are mixed: the web is a better source of reference for implementation material and systems that are constantly updated.

Then there are the VXL libraries (the Vision-*something*-Libraries, groan). This is “a collection of C++ libraries designed for computer vision research and implementation”. The CImg Library (another duff acronym: it derives from Cool Image) is a system aimed to be easy to use, efficient, and a generic base for image processing algorithms. VLFeat is “a cross-platform open source collection of vision algorithms with a special focus on visual features”. Finally, there is Southampton’s OpenIMAJ which has lots of functional capabilities and is supported by tutorials and user data (as are all the other packages). Note that these packages are open source, and there are licences and conditions on use and exploitation. Web links are shown in Table 1.2.

OpenCV	(originally Intel)	<a href="http://opencv.org/">opencv.org/</a>
VXL	Many international contributors	<a href="http://vxl.github.io">vxl.github.io</a>
CImg	Many international contributors	<a href="http://sourceforge.net/projects/cimg/">sourceforge.net/projects/cimg/</a>
VLFeat	Oxford and UCLA	<a href="http://vlfeat.org/">vlfeat.org/</a>
OpenIMAJ	Southampton	<a href="http://openimaj.org">openimaj.org</a>
<b>Table 1.2 Software packages for computer vision</b>		

### 1.5.2 Hello Python, Hello Images!

We shall be using *Python* as our main vehicle for processing images. It is now a dominant language and it is quite mature. Mark first encountered it a long time ago when we used it to arrange the conference proceedings of the British Machine Vision Conference BMVC 1998 and that was Python 1 (we also used Adobe 3.1). It has clearly moved from strength to strength, part by virtue of simplicity and part by virtue of some excellent engineering. This is not a textbook on programming in Python and for that you would best look elsewhere, e.g. [Lutz13] is a well proven text. Here we are using Python as a vehicle to show how algorithms can be made to work. For that reason, it’s not a style guide either.

Python is a scripting programming language that was developed to provide clear and simple code. It is also very general, it runs on many platforms and it has comprehensive standard libraries. Here, we use Python to show how the maths behind machine vision techniques can be translated

into working algorithms. In general, there are several levels at which you can understand a technique. It is possible to understand the basic ideas by studying the meaning and aims of a method that your program calls from a library. You can also understand a more formal definition by studying the mathematics that support the ideas. Alternatively, you can understand a technique as a series of steps that define an algorithm that processes some data. In this book, we discuss the formal ideas with mathematical terms and then present a more practical approach by including algorithms derived from the mathematical formulation.

However, presenting working algorithms is not straightforward and we have puzzled on how to code without ending up with a lot of wrapper material that can obscure functionality or whether to go to the other extreme and just end up calling a function from a package. There again, one could argue that it's best to accommodate complex code in the support material, and show its operations in the book. However, that could limit functionality, performance and complexity. Today's implementations of machine vision algorithms include concepts like multi-threading and GPU processing. Implementations follow object oriented, functional or procedural definitions that organize computations in complex and efficient data structures. They also include specific code to improve robustness. In order to keep the scope manageable and for clarity, the code in this book is limited to the main steps of algorithms. However, you should be able to understand and develop more complex implementations once you have the knowledge of the basic approach.

If you want to execute the presented code, you will need to set up Python, install some Python packages and set up the book's scripts. There are several ways to set up a Python environment and we provide setup instructions on the book's website. The scripts presented in this book do not call Python packages directly, but through two utility scripts (`ImageSupport` and `PlotSupport`) that wrap the basic image manipulations and drawing functionalities. The main idea is to present the code independently of software for drawing and loading images. You can either execute the scripts using the `ImageSupport` and `PlotSupport` definitions provided or you can implement similar utilities by using alternative Python libraries.

`ImageSupport` implements functions for reading, showing or creating images. `PlotSupport` implements functions for drawing surfaces, curves and histograms. The scripts in this book contain imports from these two utility files. In our implementation, `ImageSupport` uses `pillow` (a 'friendly' Python Imaging Library fork no less) for loading and saving images and `Numpy` to define arrays for image data. `PlotSupport` uses `Numpy` since it requires image data and it also uses `Matplotlib` for drawing. Thus, if you use the support utilities, you need to install `PIL`, `Numpy` and `Matplot` and set your environment path such that scripts can access `ImageSupport` and `PlotSupport` utilities.

```
# Feature Extraction and Image Processing
# Mark S. Nixon & Alberto S. Aguado
# Code1_1_Imagedisplay.py: Loads, inverts and displays an image.
# Shows image as a surface and prints pixel data

# Set utility folder
import sys
sys.path.insert(0, '../Utilities/')

# Set utility functions
from ImageSupport import imageReadL, showImageL, printImageRangeL, createImageF
from PlotSupport import plot3DColorHistogram

'''
Parameters:
    pathToDir = Input image directory
    imageName = Input image name
'''
pathToDir = "Input/"
imageName = "Square.png"
```

```

# Read image into array
inputImage, width, height = imageReadL(pathToDir + imageName)

# Show input image
showImageL(inputImage)

# Print pixel's values in an image range
printImageRangeL(inputImage, [0, width-1], [0, height-1])

# Create an image to store the z values for surface
outputZ = createImageF(width, height)

# Three float arrays to store colours of the surface
colorsRGB = createImageF(width, height, 3)

# Set surface and colour values
for x in range(0, width):
    for y in range(0, height):
        pixelValue = float(inputImage[y,x])
        outputZ[y,x] = pixelValue
        pointColour = float(inputImage[y,x])/255.0
        colorsRGB[y,x] = [pointColour, pointColour, pointColour]
Fig 1.1A in colour
# Plot histogram to show image
plot3DColorHistogram(outputZ, colorsRGB, [0,400])

```

### Code 1.1 Load, invert and display an image

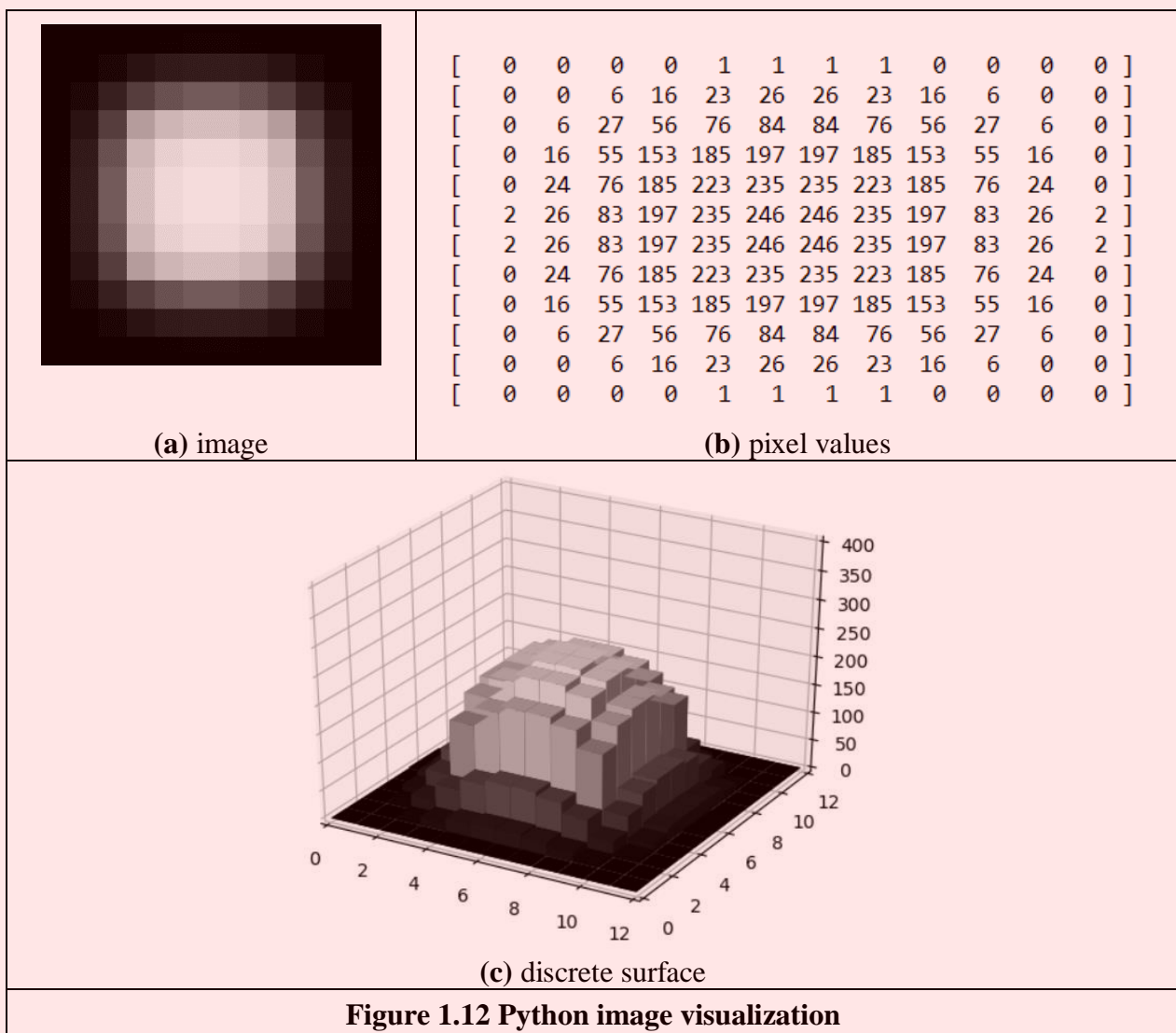
In addition to image and plot support, scripts also import functions from other utility files. This is done in the code in this book to show all the steps in an implementation algorithm whilst avoiding repetition. For example, a technique such as edge detection will be shown as a script with all the steps of the algorithm. When edge detection is used in more complex algorithms such as corner detection, the corner detection algorithm will just call a function to perform the edge detection steps. Thus, you will find the edge detection code as a script in an algorithm or as a utility function. The algorithm functions are organized by chapter, so `FourierTransformUtilities` defines functions for the scripts explained in Chapter 2 and `FeatureExtractionUtilities` for functions in Chapter 4. The code in the utilities is similar to the code explained in other parts of the book.

Code 1.1 shows our first example of a Python script. This script is used to visualize an image in different forms. It loads and displays an image, it prints the pixels' values and it shows the image as a surface. To run this code (from, say, the directory `c:\imagebook\` with the images stored in `c:\imagebook\input\`) then in the command line type `Python Code1_1_Imagedisplay.py` and it will run and produce images. The first lines of the script set the path to the directory of the utilities. Scripts in other chapters will omit these lines, but you need to set the environment in your system to access the utility files. The next part of the script imports the functions that we use for image handling and plotting. Then, we set the variables that define the parameters of the algorithm. In this case, the file name of the image we want to display. The utility function `imageReadL` reads an image and stores the data in an array (a matrix) of grey level values. The storage of image pixel data is first covered in the next Chapter, Section 2.2. The function returns an array containing a single grey level value `uint8` (8-bit unsigned integer) per pixel and the size of the image. The function `showImageL` displays an image on the screen as shown in Figure 1.13(a). In this example, the image is very small (12×12 pixels), so we can see the pixels as squares. The script uses the utility function `printImageRangeL` to display the values of the pixels in matrix form as shown in Figure 1.13(b). Note that the outer rows and



columns are nearly white with some slight variation – this variation cannot be perceived in the displayed image.

In order to display images as surfaces, the script in Code 1.1 creates images that store the height and the colour for each surface sample. The function `createImageF` creates an array of floats (floating point numbers) and its last parameter defines the number of floats per pixel. In this example, we create a one-dimensional array to store the height value of the surface and a three-dimensional array to store the colour. The arrays are filled by using the data in the input image in two nested loops. Notice that indentation is used to define the scope of each loop and that image data is accessed in in `[row][column]` format. In a grey level image, the maximum value is 255 which corresponds to white. Colour is a complex topic studied next in Section 2.2 and later in detail in Chapter 11 – we need colour here to show the images. We set the height to be larger for brighter pixels. The colour array gives the same value to the red, green and blue components, so it defines values of grey that correspond to the pixel grey value. The script uses the utility function `plot3DColorHistogram` to draw a discrete surface as shown in Figure 1.13(c).



```
# Feature Extraction and Image Processing
# Mark S. Nixon & Alberto S. Aguado
# Chapter 1: Image brightening

# Set utility folder
```

```

import sys
sys.path.insert(0, '../Utilities/')

# Iteration
from timeit import itertools

# Set utility functions
from ImageSupport import imageReadL, showImageL, printImageRangeL, createImageL

'''
Parameters:
    pathToDir = Input image directory
    imageName = Input image name
    brightDelta = Increase brightness
    printRange = Image range to print
'''
pathToDir = "Input/"
imageName = "Zebra.png"
brightDelta = 80;
printRange = [0, 10]

# Read image into array and create and output image
inputImage, width, height = imageReadL(pathToDir + imageName)
outputImage = createImageL(width, height)

# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):
    outValue = int(inputImage[y,x]) + brightDelta
    if outValue < 255:
        outputImage[y,x] = outValue
    else:
        outputImage[y,x] = 255

# Show images
showImageL(inputImage)
showImageL(outputImage)

# Print image range
printImageRangeL(inputImage, printRange, printRange)
printImageRangeL(outputImage, printRange, printRange)

```

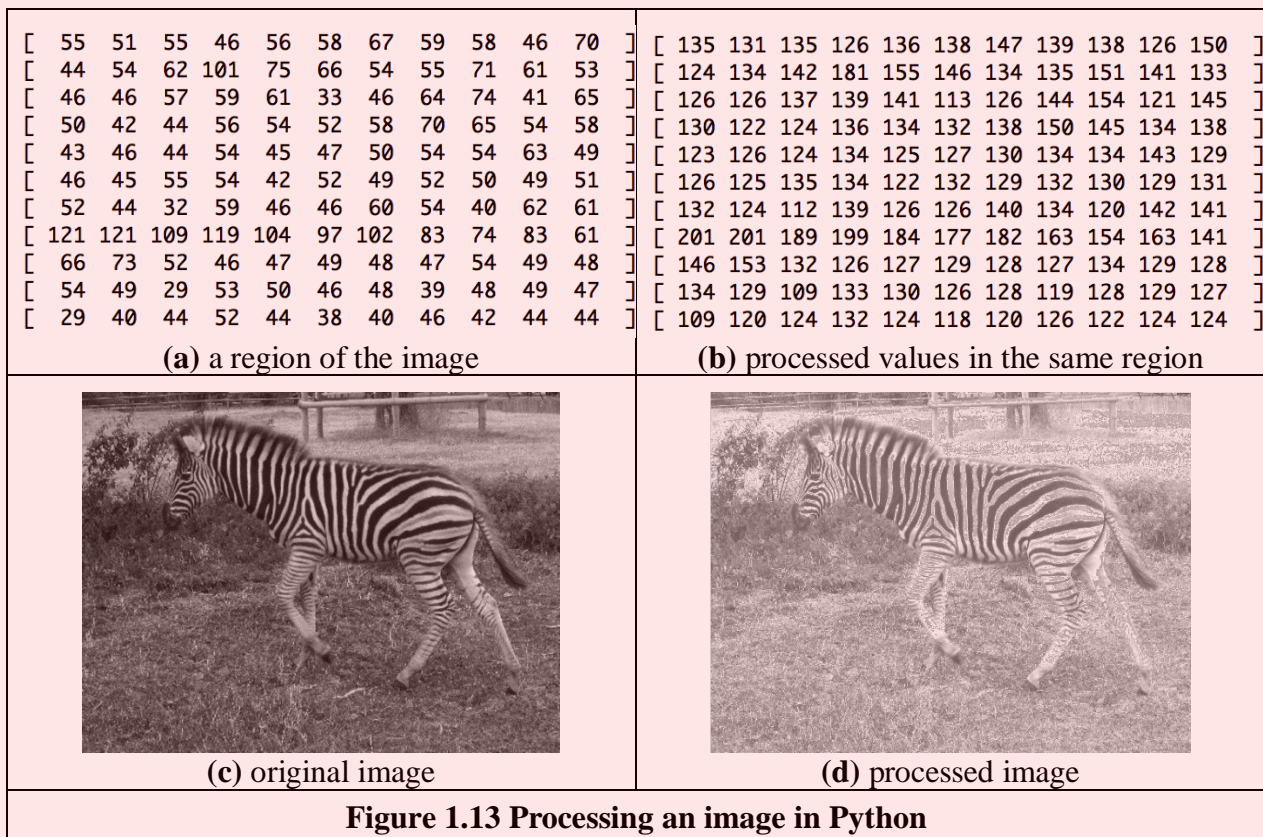
### Code 1.2 Image brightening

The script in Code 1.2 illustrates the way data in an image array is processed. The script uses the function `createImageL` to create a grey level image that contains the processed data. In addition to the name of the image, the script has two variables that are used as parameters. `brightDelta` defines an amount that is added to the input pixel to produce an output pixel and `printRange` defines a range of pixels that are printed out. Similar to Code 1.1, the script iterates through all the values of the input image by addressing the columns from 0 to width, and the rows from 0 to height. However, Code 1.2 defines the two nested loops in a single line by using the function `itertools.product` imported from `timeit`. Both the iterative and the nested loops have the same functionality. This book uses the iterator to perform nested loops to reduce indentation and keep the code clearer. Inside the loop, the output pixels  $q(x, y)$  result from adding  $\Delta_{bright}$  to some input pixels  $p(x, y)$  as

$$q(x, y) = p(x, y) + \Delta_{bright} \quad (1.1)$$

In code, this is computed as the value of the input pixel plus the value of `brightDelta`. There is some jacketing software which prevents values exceeding 255 – if the result of addition exceeds this then the value stops at 255. This is one large difference between describing algorithms as maths

and as code – some of the code never appears in the maths but is necessary to make the maths work correctly. Figure 1.14 shows the result of the script. The figure shows the input and the resulting image as well as some of the pixel values in matrix form.



### 1.5.3 Mathematical Tools

Mathematica, Maple and Matlab are amongst the most popular of current mathematical systems, and Mathcad was used in earlier editions of this book. There have been surveys that compare their efficacy, but it is difficult to ensure precise comparison due to the impressive speed of development of techniques. Most systems have their protagonists and detractors, as in any commercial system. There are many books which use these packages for particular subjects, and there are often handbooks as addenda to the packages. We shall use Matlab throughout this text, aiming to expose the range of systems that are available. Matlab dominates this market these days, especially in image processing and computer vision, and its growth rate has been enormous; older versions of this text used Mathcad which was more sophisticated (and WYSWYG) but had a more chequered commercial history. Note that there is an open source compatible system for Matlab called Octave. This is of course free. Most universities have a site license for Matlab and without that it can be rather (too) expensive for many. The web links for the main mathematical packages are given in Table 1.3.

General		
Guide to available Mathematical Software	NIST	<a href="http://gams.nist.gov/">gams.nist.gov/</a>

Vendors		
Mathcad	Parametric Technology Corp.	<a href="http://ptc.com/en/products/Mathcad">ptc.com/en/products/Mathcad</a>
Mathematica	Wolfram Research	<a href="http://wolfram.com/">wolfram.com/</a>
Matlab	Mathworks	<a href="http://mathworks.com/">mathworks.com/</a>
Maple	Maplesoft	<a href="http://maplesoft.com/">maplesoft.com/</a>
Matlab Compatible		
Octave	Gnu (Free Software Foundation)	<a href="http://gnu.org/software/octave/">gnu.org/software/octave/</a>
<b>Table 1.3 Mathematical package websites</b>		

### 1.5.4 Hello Matlab

For those who are less enamoured with programming we shall use the *Matlab* package to accompany Python. Matlab offers a set of mathematical tools and visualisation capabilities in a manner arranged to be very similar to conventional computer programs. The system was originally developed for matrix functions, hence the 'Mat' in the name. There are a number of advantages to Matlab, not least the potential speed advantage in computation and the facility for debugging, together with a considerable amount of established support. There is an image processing toolkit supporting Matlab, but it is rather limited compared with the range of techniques exposed in this text. Matlab's popularity is reflected in a book dedicated to its use for image processing [Gonzalez09], by perhaps one of the subject's most popular authors. It is of note that many researchers make available Matlab versions for others to benefit from their new techniques.

The Octave open source compatible system was built mainly with Matlab compatibility in mind. It shares a lot of features with Matlab such as using matrices as the basic data type, with availability of complex numbers and built in functions as well the capability for user-defined functions. There are some differences between Octave and Matlab, and there is extensive support for both. We shall refer to both systems using Matlab as a general term.

Essentially, Matlab is the set of instructions that process the data stored in a workspace, which can be extended by user-written commands. The workspace stores the different lists of data and these data can be stored in a MAT file; the user-written commands are functions that are stored in M-files (files with extension .M). The procedure operates by instructions at the command line to process the workspace data using either one of Matlab's own commands, or using your own commands. The results can be visualised as graphs, surfaces or as images.

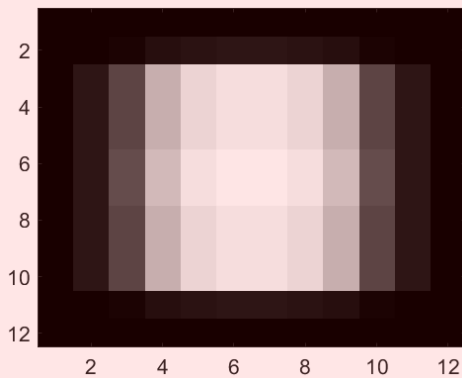
Matlab provides powerful matrix manipulations to develop and test complex implementations. In this book, we avoid matrix implementations in favour of a more C++ algorithmic form. Thus, matrix expressions are transformed into loop sequences. This helps students without experience in matrix algebra to understand and implement the techniques without dependency on matrix manipulation software libraries. Implementations in this book only serve to gain understanding of the techniques' performance and correctness, and favour clarity rather than speed.

Matlab processes images, so we need to know what an image represents. Images are spatial data, which is data that is indexed by two spatial co-ordinates. The camera senses the brightness at a point with co-ordinates  $x,y$ . Usually,  $x$  and  $y$  refer to the horizontal and vertical axes, respectively. Throughout this text we shall work in orthographic projection, ignoring perspective, where real world co-ordinates map directly to  $x$  and  $y$  co-ordinates in an image. The homogeneous co-ordinate system is a popular and proven method for handling three-dimensional co-ordinate systems ( $x$ ,  $y$  and  $z$  where  $z$  is depth), though it will not be used here as we shall process two-dimensional images

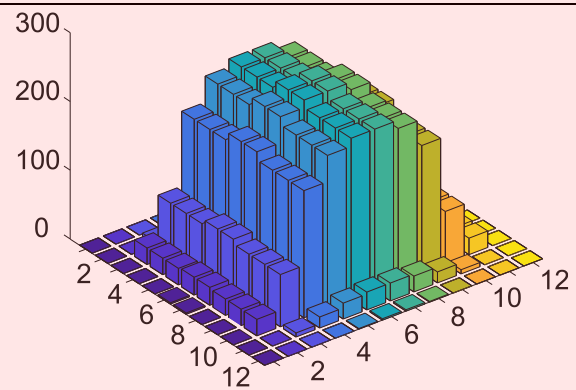
only. The brightness sensed by a camera is transformed to a signal which is and stored as a value within the computer, referenced to the co-ordinates x,y in the image. Accordingly, a computer image is a matrix of points. For a greyscale image, the value of each point is proportional to the brightness of the corresponding point in the scene viewed by the camera. These points are the picture elements, or pixels.

0	0	0	0	1	2	2	1	0	0	0	0
0	0	6	16	23	26	26	23	16	6	0	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
2	26	83	197	235	246	246	235	197	83	26	2
2	26	83	197	235	246	246	235	197	83	26	2
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	24	76	185	223	235	235	223	185	76	24	0
0	0	6	16	23	26	26	23	16	6	0	0
0	0	0	0	1	2	2	1	0	0	0	0

(a) set of pixel values



(b) Matlab image



(c) Matlab surface plot

Figure 1.14 Matlab image visualisation

```

%read in an image
a=imread('Input\Square.png');
%view image
imagesc(a);
%view it in black and white
disp ('We shall display the data as a grey level image')
colormap(gray);
% Let's hold so we can view it
disp ('When you are ready to move on, press RETURN')
    
```

```

pause;
disp ('Now we shall display the data as a surface')
bar3(a);

```

### Code 1.3 Example Matlab script

Consider for example, the set of pixel values in Figure 1.14(a). These values were derived from the image of a bright peak on a dark background. The peak is brighter where the pixels have a larger value (here over 200 brightness levels); the background is dark and those pixels have a smaller value (near 0 brightness levels). (We shall consider how many points we need in an image and the possible range of values for pixels in Chapter 2.) The peak can be viewed as an image in Figure 1.14(b), or as a surface in Figure 1.14(c).

As in the Python code, there are considerably more sophisticated systems and usage than this version of Matlab code. Like Python in Section 1.5.2, our aim here is not to show people how to use Matlab and all its wonderful functionality. We are using the simplest versions with default settings so that people can process images quickly and conveniently. The Matlab code might appear clearer than the Python code. It is, but that is at the cost of access to data and Python has much greater freedom there. One can get to Manchester from London using a Mini or a Rolls Royce: the style is different (and the effect on the environment is different...) but the result is the same. That is how we shall use Python and Matlab in this textbook.

Matlab runs on Unix/Linux, Windows and Macintosh systems; a student version is available. We shall use a script, to develop our approaches, which is the simplest type of M-file, as illustrated in Code 1.3. To start the Matlab system, type `MATLAB` at the command line. At the Matlab prompt (`>>`) type `chapter1` to load and run the script (given that the file `Code_1_3.m` is saved in the directory you are working in). Alternatively, open the directory containing `Code_1_3.m` and click on it to run it. Comments are preceded by a `%`; we first read in an image. Having set the display facility to black and white, we can view the array of points `a` as a surface. When the image, illustrated in Figure 1.14(b), has been plotted, then Matlab has been made to `pause` until you press Return before moving on. Here, when you press Return, you will next see the surface of the array, Figure 1.14(c).

```

function inverted = invert(image)

% Subtract image point brightness from maximum
% Usage: new image = invert(old image)
% Parameters: image      - array of points

[rows,cols]=size(image); % get dimensions

maxi = max(max(image)); % find the maximum

% subtract image points from maximum
for x = 1:cols %address all columns
    for y = 1:rows %address all rows
        inverted(y,x)=maxi-image(y,x);
    end
end

```

### Code 1.4 Matlab function `invert.m` to invert an Image

We can use Matlab's own command to interrogate the data: these commands find use in the M-files that store subroutines. An example routine is called `invert`. This subroutine is stored in a file called `invert.m` and is a function that inverts brightness by subtracting the value of each point from the array's maximum value. The code is illustrated in Code 1.4. Note that this code uses for loops which are best avoided to improve speed, using Matlab's vectorised operations. The whole procedure can actually be implemented by the command `inverted = max(max(pic)) - pic`. In fact, one of Matlab's assets is a 'profiler' which allows you to determine exactly how much time is spent on different parts of your programs. Naturally, there is facility for importing graphics files, which is quite extensive (i.e., it accepts a wide range of file formats). When images are used we find that Matlab has a range of datatypes. We must move from the unsigned integer datatype, used for images, to the double precision datatype to allow processing as a set of real numbers. (Matlab does make this rather easy, and can handle complex numbers with ease too.) In these ways Matlab can, and will be used to process images throughout this book, with the caveat that this is not a book about Matlab (or Python). Note that the translation to application code is perhaps easier via Matlab than for other systems and it offers direct compilation of the code. There are some Matlab scripts available at the book's website (<https://www.southampton.ac.uk/~msn/book/>) for on-line tutorial support of the material in this book. As with the caveat earlier, these are provided for educational purposes only. There are many other implementations of techniques available on the Web in Matlab. The edits required to make the Matlab worksheets run in Octave are described in the file `readme.txt` in the downloaded zip.

## 1.6 Associated Literature

### 1.6.1 Journals, Magazines and Conferences

As in any academic subject, there are many sources of literature and when used within this text the cited references are to be found at the end of each chapter. The professional magazines include those that are more systems oriented, like *Vision Systems Design*. These provide more general articles, and are often a good source of information about new computer vision products. For example, they survey available equipment, such as cameras and monitors, and provide listings of those available, including some of the factors by which you might choose to purchase them.

There is a wide selection of research journals - probably more than you can find in your nearest library unless it is particularly well-stocked. These journals have different merits: some are targeted at short papers only, whereas some have short and long papers; some are more dedicated to the development of new theory whereas others are more pragmatic and focus more on practical, working, image processing systems. But it is rather naive to classify journals in this way, since all journals welcome good research, with new ideas, which has been demonstrated to satisfy promising objectives.

The main research journals include: *IEEE Transactions on: Pattern Analysis and Machine Intelligence* (in later references this will be abbreviated to *IEEE Trans. on PAMI*); *Image Processing (IP)*; *Systems, Man and Cybernetics (SMC)*; and on *Medical Imaging* (there are many more IEEE transactions, e.g. the *IEEE Transactions on Biometrics, Behavior and Identity Science*, which sometimes publish papers of interest in, or application of, image processing and computer vision). The *IEEE Transactions* are usually found in (university) libraries since they are available at comparatively low cost; they are online to subscribers at the IEEE Explore site ([ieeexplore.ieee.org/](http://ieeexplore.ieee.org/)) and this includes conferences. *Computer Vision and Image Understanding* and *Graphical Models and Image Processing* arose from the splitting of one of the subject's earlier journals, *Computer Vision, Graphics and Image Processing (CVGIP)*, into two parts. Do not confuse *Pattern Recognition (Pattern Recog.)* with *Pattern Recognition Letters (Pattern Recog. Lett.)*, published under the aegis of the Pattern Recognition Society and the International

Association of Pattern Recognition, respectively, since the latter contains shorter papers only. The *International Journal of Computer Vision* and *Image and Vision Computing* are well established journals too. Finally, do not miss out on the *IET Computer Vision* and other journals.

Most journals are now on-line but usually to subscribers only; some go back a long way. Academic Press titles include *Computer Vision and Image Understanding*, *Graphical Models and Image Processing* and *Real-Time Image Processing*.

There are plenty of *conferences* too: the proceedings of IEEE conferences are also held on the Explore site and two of the top conferences are *Computer Vision and Pattern Recognition (CVPR)* which is held annually in the USA, the *International Conference on Computer Vision (ICCV)* is biennial and moves internationally. The IEEE also hosts specialist conferences, e.g. on biometrics or on computational photography. *Lecture Notes in Computer Science* are hosted by Springer ([springer.com](http://springer.com)) and are usually the proceedings of conferences. Some conferences such as the *British Machine Vision Conference* series maintain their own site ([bmva.org](http://bmva.org)). The excellent Computer Vision Conferences page [conferences.visionbib.com/Iris-Conferences.html](http://conferences.visionbib.com/Iris-Conferences.html) is brought to us by Keith Price and lists conferences in Computer Vision, Image Processing and Pattern Recognition.

### 1.6.2 Textbooks

There are many *textbooks* in this area. Increasingly, there are web versions, or web support, as summarised in Table 1.4. The difficulty is of access as you need a subscription to be able to access the online book (and sometimes even to see that it is available online), though there are also Kindle versions. For example, this book is available online to those subscribing to Referex in Engineering Village [engineeringvillage.org](http://engineeringvillage.org). The site given in Table 1.4 for this book is the support site which includes demonstrations, worksheets, errata and other information. The site given next, at Edinburgh University UK, is part of the excellent CVOnline site (many thanks to Bob Fisher there) and it lists current books as well pdfs of which some are more dated, but still excellent (e.g. [Ballard82]). There is also continuing debate on appropriate education in image processing and computer vision, for which review material is available [Bebis03].

For support material, the *World of Mathematics* comes from Wolfram research (the distributors of Mathematica) and gives an excellent web based reference for mathematics. *Numerical Recipes* [Press07] is one of the best established texts in signal processing. It is beautifully written, with examples and implementation and is on the web too. *Digital Signal Processing* is an online site with focus on the more theoretical aspects which will be covered in Chapter 2. *The Joy of Visual Perception* is an online site on how the human vision system works. We haven't noted *Wikipedia* – computer vision is there too.

By way of context, for comparison with other textbooks, this text aims to start at the foundation of computer vision, and to reach current research. Its content specifically addresses techniques for image analysis, considering feature extraction and shape analysis in particular. Matlab and Python are used as vehicles to demonstrate implementation. There are of course other texts, and these can help you to develop your interest in other areas of computer vision.

This book's homepage	Southampton University	<a href="https://www.southampton.ac.uk/~msn/book/">https://www.southampton.ac.uk/~msn/book/</a>
CVOnline – online book compendium	Edinburgh University	<a href="http://homepages.inf.ed.ac.uk/rbf/CVonline/books.htm">homepages.inf.ed.ac.uk/rbf/CVonline/books.htm</a>
World of Mathematics	Wolfram Research	<a href="http://mathworld.wolfram.com">mathworld.wolfram.com</a>



Numerical Recipes	Cambridge University Press	<a href="http://numerical.recipes">numerical.recipes</a>
Digital Signal Processing	Steven W. Smith	<a href="http://dspguide.com">dspguide.com</a>
The Joy of Visual Perception	York University	<a href="http://www.yorku.ca/research/vision/eye/thejoy.htm">http://www.yorku.ca/research/vision/eye/thejoy.htm</a>
<b>Table 1.4 Web textbooks and homepages</b>		

Some of the main textbooks are now out of print, but pdfs can be found at the CVOnline site. There are more than given here, some of which will be referred to in later Chapters; each offers a particular view or insight into computer vision and image processing. Some of the main textbooks include: *Vision* [Marr82] which concerns vision and visual perception (as previously mentioned); *Fundamentals of Computer Vision* [Jain89] which is stacked with theory and technique, but omits implementation and some image analysis, as does *Robot Vision* [Horn86]; *Machine Vision* [Jain95] offers concise coverage of 3D and motion; *Digital Image Processing* [Gonzalez17] has more tutorial element than many of the basically theoretical texts and has a fantastic reputation for introducing people to the field; *Digital Picture Processing* [Rosenfeld82] is very dated now, but is a well-proven text for much of the basic material; and *Digital Image Processing* [Pratt01], which was originally one of the earliest books on image processing and, like *Digital Picture Processing*, is a well-proven text for much of the basic material, particularly image transforms. Despite its name, *Active Contours* [Blake98] concentrates rather more on models of motion and deformation and probabilistic treatment of shape and motion, than on the active contours which we shall find here. As such it is a more research text, reviewing many of the advanced techniques to describe shapes and their motion. *Image Processing - The Fundamentals* [Petrou10] (by two Petrous!) surveys the subject (as its title implies) from an image processing viewpoint. *Computer Vision* [Shapiro01] includes chapters on image databases and on virtual and augmented reality. *Computer Vision: A Modern Approach* [Forsyth11] offers much new – and needed – insight into this continually developing subject (two chapters that didn't make the final text – on probability and on tracking – are available at the book's website [luthuli.cs.uiuc.edu/~daf/book/book.html](http://luthuli.cs.uiuc.edu/~daf/book/book.html)). One text [Brunelli09] focusses on object recognition techniques “employing the idea of projection to match image patterns” which is a class of approaches to be found later in this text. A much newer text *Computer Vision: Algorithms and Applications* [Szeliski11] is naturally much more up to date than older texts, and has an online (earlier) electronic version available too. An excellent (and popular) text *Computer Vision Models, Learning, and Inference* [Prince12] – electronic version available – is based on models and learning. One of the bases of the book is “to organize our knowledge ... what is most critical is the model itself - the statistical relationship between the world and the measurements” and thus covers many of the learning aspects of computer vision which complement and extend this book's focus on feature extraction. Finally, very recently there has been a condensed coverage of Image Processing and Analysis [Gimel'farb18].

Also, Kasturi, R., and Jain, R. C. Eds.: *Computer Vision: Principles* [Kasturi91a] and *Computer Vision: Advances and Applications* [Kasturi91b] presents a collection of seminal papers in computer vision, many of which are cited in their original form (rather than in this volume) in later chapters. There are other interesting edited collections [Chellappa92], one edition [Bowyer96] honours Azriel Rosenfeld's many contributions.

Section 1.5 describes some of the image processing software packages available, and their textbook descriptions. Of the texts with a more practical flavour *Image Processing and Computer Vision* [Parker10] includes description of software rather at the expense of lacking range of technique. There is excellent coverage of practicality in *Practical Algorithms for Image Analysis*

[O’Gorman08]. One JAVA text, *The Art of Image Processing with Java*, [Hunt11] emphasises software engineering more than feature extraction (giving basic methods only).

Other textbooks include: the longstanding *The Image Processing Handbook* [Russ17] which contains much basic technique with excellent visual support, but without any supporting theory; *Computer Vision: Principles, Algorithms, Applications, Learning* [Davies17] which is targeted primarily at (industrial) machine vision systems but covers much basic technique, with pseudo-code to describe their implementation; and the *Handbook of Pattern Recognition and Computer Vision* [Cheng16] covers much technique. There are specialist texts too and they usually concern particular Sections of this book, and they will be mentioned there. Last but by no means least, there is even a (illustrated) dictionary [Fisher13] to guide you through the terms that are used.

### 1.6.3 The Web

This book's homepage ([users.soton.ac.uk/msn/book](http://users.soton.ac.uk/msn/book)) details much of the support material, including worksheets, code and demonstrations, and any errata we regret have occurred (and been found). The Computer Vision Online CVOnline homepage [dai.ed.ac.uk/CVonline/](http://dai.ed.ac.uk/CVonline/) has been brought to us by Bob Fisher from the University of Edinburgh. There's a host of material there, including its description. If you have access to the web-based indexes of published papers, the Web of Science covers a lot and includes INSPEC which has papers more related to engineering, together with papers in conferences. Explore is for the IEEE – with paper access for subscribers; many researchers turn to Google Scholar as this is freely available with ability to retrieve the papers as well as to see where they have been used. Many use Scholar since it also allows you to see the collection (and metrics) of papers in authors' pages. The availability of papers within these systems has been changing with the new Open Access systems, which at the time of writing are still in some flux.

## 1.7 Conclusions

This Chapter has covered most of the pre-requisites for feature extraction in image processing and computer vision. We need to know how we see, in some form, where we can find information and how to process data. More importantly we need an image, or some form of spatial data. This is to be stored in a computer and processed by our new techniques. As it consists of data points stored in a computer, this data is sampled or discrete. Extra material on image formation, camera models and image geometry is to be found in Chapter 10, but we shall be considering images as a planar array of points hereon. We need to know some of the bounds on the sampling process, on how the image is formed. These are the subjects of the next Chapter which also introduces a new way of looking at the data, how it is interpreted (and processed) in terms of frequency.

## 1.8 Chapter 1 References

- [Adelson05] Adelson, E. H. and Wang, J. Y. A., 1992. Single lens stereo with a plenoptic camera. *IEEE Trans. on PAMI*, (2), pp 99-106
- [Armstrong91] Armstrong, T., *Colour Perception - A Practical Approach to Colour Theory*, Tarquin Publications, Diss UK, 1991
- [Ballard82] Ballard, D. H., Brown, C. M., *Computer Vision*, Prentice-Hall Inc New Jersey USA, 1982
- [Bebis03] Bebis, G., Egbert, D., and Shah, M., Review of Computer Vision Education, *IEEE Transactions on Education*, **46**(1), pp. 2-21, 2003
- [Blake98] Blake, A., and Isard, M., *Active Contours*, Springer-Verlag London Limited, London UK, 1998

- [Bowyer96] Bowyer, K., and Ahuja, N., Eds., *Advances in Image Understanding, A Festschrift for Aziel Rosenfeld*, IEEE Computer Society Press, Los Alamitos, CA USA, 1996
- [Bradski08] Bradski, G., and Kaehler, A., *Learning OpenCV: Computer Vision with the OpenCV Library*, O'Reilly Media, Inc, Sebastopol, CA USA, 2008
- [Bruce90] Bruce, V., and Green, P., *Visual Perception: Physiology, Psychology and Ecology*, 2<sup>nd</sup> Edition, Lawrence Erlbaum Associates, Hove UK, 1990
- [Chellappa92] Chellappa, R., *Digital Image Processing*, 2nd Edition, IEEE Computer Society Press, Los Alamitos, CA USA, 1992
- [Cheng16] Cheng, C. H., and Wang, P. S. P., *Handbook of Pattern Recognition and Computer Vision*, 5<sup>th</sup> Edition, World Scientific, Singapore, 2016
- [Cornsweet70] Cornsweet, T. N., *Visual Perception*, Academic Press Inc., N.Y. USA, 1970
- [Davies17] Davies, E. R., *Computer Vision: Principles, Algorithms, Applications, Learning*, Academic Press, 5<sup>th</sup> Edition, 2017
- [Fisher2013] Fisher, R. B., Breckon, T. P., Dawson-Howe, K., Fitzgibbon, A., and Robertson, C., Trucco, E., and Williams C. K. I., *Dictionary of Computer Vision and Image Processing*, Wiley, Chichester UK, 2<sup>nd</sup> Ed 2013
- [Forsyth11] Forsyth, D., and Ponce, J., *Computer Vision: A Modern Approach*, Prentice Hall, N.J. USA, 2<sup>nd</sup> Edition 2011
- [Fossum97] Fossum, E. R., CMOS Image Sensors: Electronic Camera-On-A-Chip, *IEEE Trans. Electron Devices*, **44**(10), pp1689-1698, 1997
- [Gimel'farb18] Gimel'farb, G., Patrice, D., *Image Processing and Analysis: A Primer*, World Scientific Europe Ltd, London UK, 2018
- [Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4<sup>th</sup> Edition, Pearson Education, 2017
- [Gonzalez09] Gonzalez, R. C., Woods, R. E., and Eddins, S. L., *Digital Image Processing using MATLAB*, Prentice Hall, 2<sup>nd</sup> Edition, 2009
- [Horn86] Horn, B. K. P., *Robot Vision*, MIT Press, Boston USA, 1986
- [Hunt11] Hunt, K. A., *The Art of Image Processing with Java*, CRC Press (A. K. Peters Ltd.), Natick MA USA, 2011
- [Jain89] Jain A. K., *Fundamentals of Computer Vision*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 1989
- [Jain95] Jain, R. C., Kasturi, R., and Schunk, B. G., *Machine Vision*, McGraw-Hill Book Co., Singapore, 1995
- [Kasturi91a] Kasturi, R., and Jain, R. C., *Computer Vision: Principles*, IEEE Computer Society Press, Los Alamitos CA USA, 1991
- [Kasturi91b] Kasturi, R., and Jain, R. C., *Computer Vision: Advances and Applications*, IEEE Computer Society Press, Los Alamitos CA USA, 1991
- [Kuroda14] Kuroda, T., *Essential Principles of Image Sensors*, CRC Press, Boca Raton FL USA, 2014
- [Lichtsteiner08] Lichtsteiner, P., Posch, C., and Delbruck, T., A 128×128 120 db 15µs latency asynchronous temporal contrast vision sensor. *IEEE J. Solid-State Circuits*, **43**(2), pp 566–576, 2008
- [Livingstone14] Livingstone, M. S., *Vision and Art (Updated and Expanded Edition)*, Harry N. Abrams, New York USA, 2<sup>nd</sup> Edition 2014
- [Lutz13] Lutz, M., *Learning Python*, O'Reilly Media, Sebastopol CA USA, 5<sup>th</sup> Edition 2013

- [Marr82] Marr, D., *Vision*, W. H. Freeman and Co., N.Y. USA, 1982
- [Nakamura05] Nakamura, J., *Image Sensors and Signal Processing for Digital Still Cameras*, CRC Press, Boca Raton FL USA, 2005
- [O’Gorman08] O’Gorman, L., and Sammon, M. J., Seoul, M., *Practical Algorithms for Image Analysis*, 2<sup>nd</sup> Edition, Cambridge University Press, Cambridge UK, 2008
- [Overington92] Overington, I., *Computer Vision - A Unified, Biologically-Inspired Approach*, Elsevier Science Press, Holland, 1992
- [Parker10] Parker, J. R., *Algorithms for Image Processing and Computer Vision*, Wiley, Indianapolis IN USA, 2<sup>nd</sup> Edition 2010
- [Petrou10] Petrou, M., and Petrou, C., *Image Processing - The Fundamentals*, Wiley-Blackwell, London UK, 2<sup>nd</sup> Edition, 2010
- [Phillips18] Phillips J. B., and Eliasson, H., *Camera Image Quality Benchmarking*, Wiley, Hoboken NJ USA, 2018
- [Pratt01] Pratt, W. K., *Digital Image Processing: PIKS Inside*, Wiley, 3<sup>rd</sup> Edition, 2001
- [Press07] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes 3<sup>rd</sup> Edition: The Art of Scientific Computing*, Cambridge University Press, Cambridge UK, 3<sup>rd</sup> Edition, 2007
- [Prince12] Prince, S. J. D., *Computer Vision Models, Learning, and Inference*, Cambridge University Press, Cambridge UK, 2012
- [Ratliff65] Ratliff, F., *Mach Bands: Quantitative Studies on Neural Networks in the Retina*, Holden-Day Inc., San Francisco USA, 1965
- [Rosenfeld82] Rosenfeld, A., and Kak, A. C., *Digital Picture Processing*, 2<sup>nd</sup> Edition, Vols. 1 and 2, Academic Press Inc., Orlando FL USA, 1982
- [Russ17] Russ, J. C., *The Image Processing Handbook*, 7<sup>th</sup> Edition, CRC Press (Taylor & Francis), Boca Raton FL USA, 2017
- [Shapiro01] Shapiro, L. G., and Stockman, G. C., *Computer Vision*, Prentice Hall, 2001
- [Solem12] Solem, JE, *Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images*, O’Reilly Media Newton Mass. USA, 2012
- [Son17] Son, B., Suh, Y., Kim, S., Jung, H., Kim, J.S., Shin, C., Park, K., Lee, K., Park, J., Woo, J. and Roh, Y., A 640×480 dynamic vision sensor with a 9µm pixel and 300Meps address-event representation. *Proc. 2017 IEEE Int.l Solid-State Circuits Conf. (ISSCC)*. pp 66-67. 2017
- [Szeliski11] Szeliski, R., *Computer Vision: Algorithms and Applications*, Springer Verlag, London UK, 2011
- [Wu17] Wu, G., Masia, B., Jarabo, A., Zhang, Y., Wang, L., Dai, Q., Chai, T. and Liu, Y., Light field image processing: an overview. *IEEE J. of Selected Topics in Signal Processing*, **11**(7), pp 926-954, 2017

## 2 Images, Sampling and Frequency Domain Processing

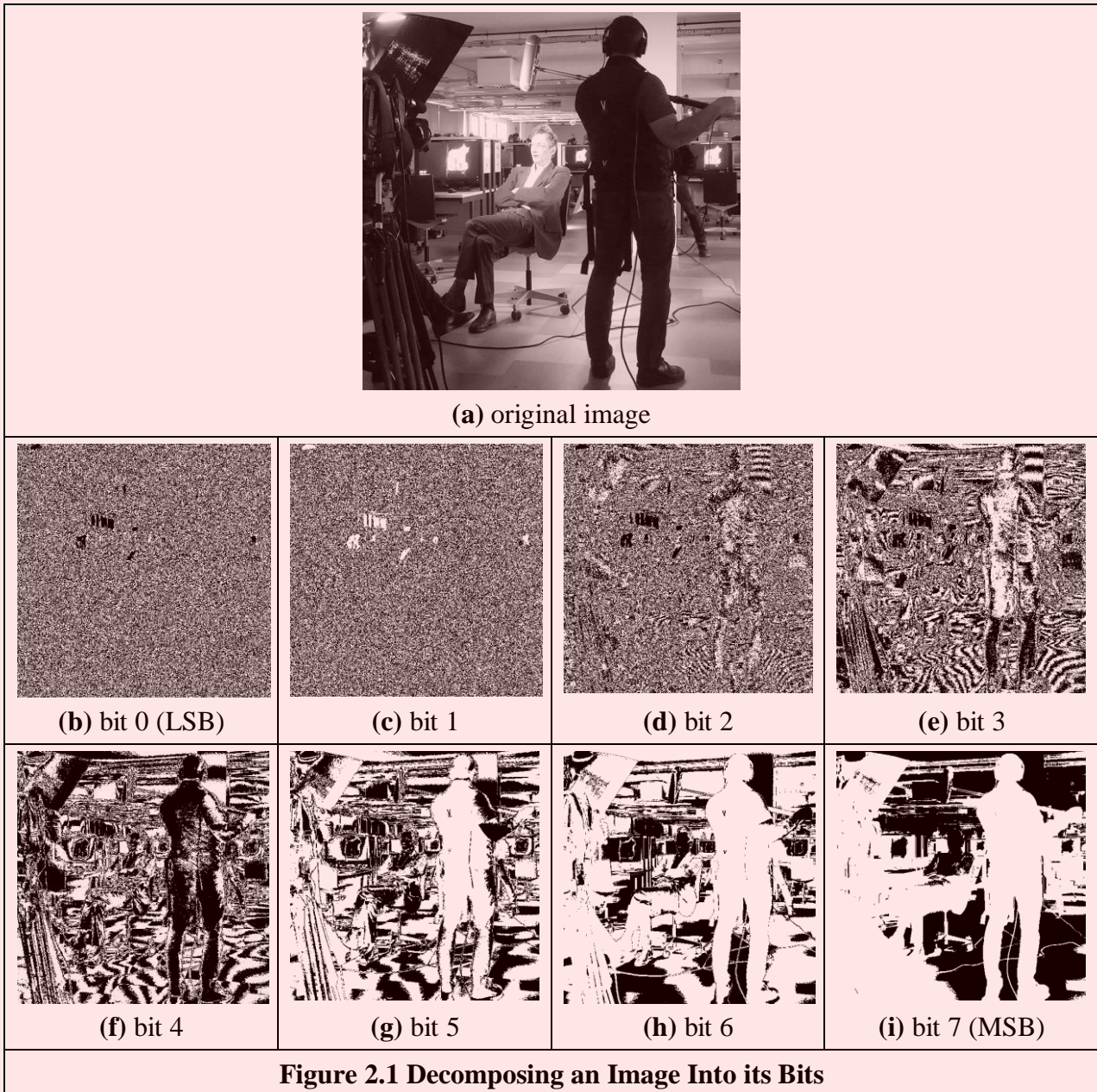
### 2.1 Overview

In this chapter, we shall look at the basic theory which underlies image formation and processing. We shall start by investigating what makes up a picture and look at the consequences of having a different number of points in the image. We shall also look at images in a different representation, known as the frequency domain. In this, as the name implies, we consider an image as a collection of frequency components. We can actually operate on images in the frequency domain and we shall also consider different transformation processes. These allow us different insights into images and image processing which will be used in later chapters not only as a means to develop techniques, but also to give faster (computer) processing.

Main topic	Sub topics	Main points
Images	Effects of differing numbers of points and of number range for those points.	<i>Grayscale, colour, resolution, dynamic range, storage.</i>
Fourier transform theory	What is meant by the frequency domain, how it applies to discrete (sampled) images, how it allows us to interpret images and the sampling resolution (number of points).	<i>Continuous Fourier transform and properties, sampling criterion, discrete Fourier transform and properties, image transformation, transform duals. Inverse Fourier transform. Importance of magnitude and phase.</i>
Consequences of transform approach	Basic properties of Fourier transforms, other transforms, frequency domain operations.	<i>Translation (shift), rotation and scaling. Principle of Superposition and linearity. Walsh, Hartley, Discrete Cosine and Wavelet transforms. Filtering and other operations.</i>
<b>Table 2.1 Overview of Chapter 2</b>		

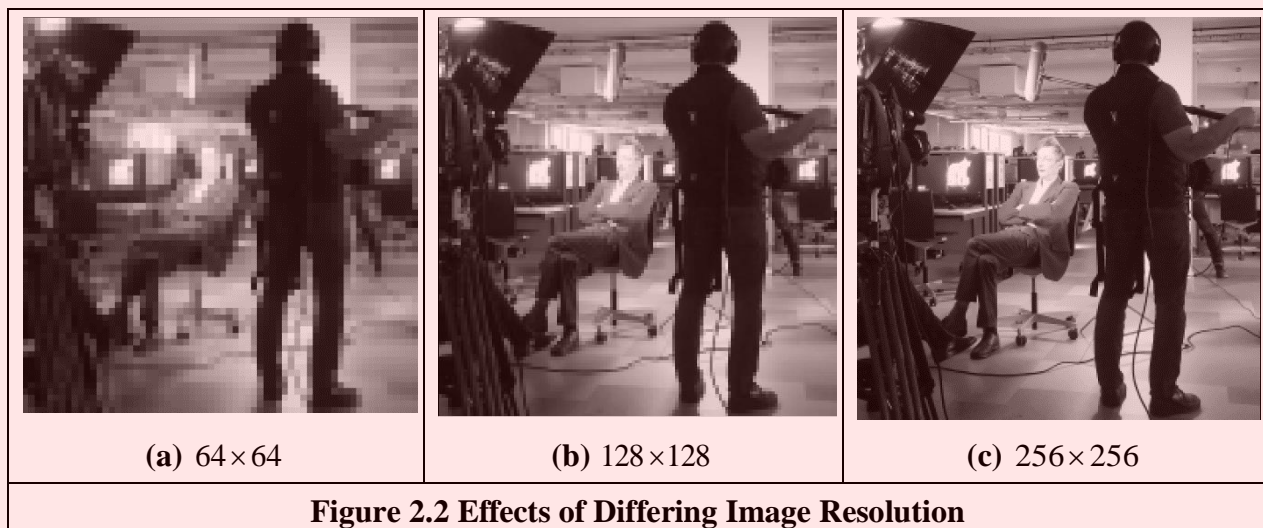
### 2.2 Image Formation

A computer image is a matrix (a two dimensional array) of *pixels*. The value of each pixel is proportional to the brightness of the corresponding point in the scene; its value is, of course, usually derived from the output of an A/D converter. We can define a square image as  $N \times N$   $m$ -bit pixels where  $N$  is the number of points and  $m$  controls the number of brightness values. Using  $m$  bits gives a range of  $2^m$  values, ranging from 0 to  $2^m - 1$ . If  $m$  is 8 this gives brightness levels ranging between 0 and 255, which are usually displayed as black and white, respectively, with shades of grey in-between, as they are for the *grayscale image* of a scene in Figure 2.1(a). Smaller values of  $m$  give fewer available levels reducing the available contrast in an image. We are concerned with images here, not their formation; imaging geometry (pinhole cameras et al) is to be found in Chapter 10.



The ideal value of  $m$  is actually related to the *signal to noise ratio (dynamic range)* of the camera. This is stated as approximately 45 dB for an analogue camera and since there are 6 dB per bit, then 8 bits will cover the available range. Choosing 8-bit pixels has further advantages in that it is very convenient to store pixel values as bytes, and 8-bit A/D converters are cheaper than those with a higher resolution. For these reasons images are nearly always stored as 8-bit bytes, though some applications use a different range. These are the 8-bit numbers encountered in Section 1.5.2, stored as unsigned 8-bit bytes (`uint8`). The relative influence of the eight bits is shown in the image of the subjects in Figure 2.1. Here, the least significant bit, bit 0 (Figure 2.1(b)), carries the least information (it changes most rapidly) and is largely noise. As the order of the bits increases, they change less rapidly and carry more information. The most information is carried by the most significant bit, bit 7 (Figure 2.1(i)). Clearly, the fact that there are people in the original image can be recognised much better from the high order bits, much more reliably than it can from the other bits (also notice the odd effects which would appear to come from lighting in the middle order bits). The variation in lighting is hardly perceived by human vision.

*Colour images* (also mentioned in Section 1.5.2) follow a similar storage strategy to specify pixels' intensities. However, instead of using just one image plane, colour images are represented by three intensity components. These components generally correspond to red, green, and blue (the RGB model) although there are other colour schemes. For example, the CMYK colour model is defined by the components cyan, magenta, yellow, and black. In any colour mode, the pixel's colour can be specified in two main ways. First, you can associate an integer value, with each pixel, that can be used as an index to a table that stores the intensity of each colour component. The index is used to recover the actual colour from the table when the pixel is going to be displayed, or processed. In this scheme, the table is known as the image's palette and the display is said to be performed by colour mapping. The main reason for using this colour representation is to reduce memory requirements. That is, we only store a single image plane (i.e., the indices) and the palette. This is less than storing the red, green and blue components separately and so makes the hardware cheaper and it can have other advantages, for example when the image is transmitted. The main disadvantage is that the quality of the image is reduced since only a reduced collection of colours is actually used. An alternative to represent colour is to use several image planes to store the colour components of each pixel. This scheme is known as true colour and it represents an image more accurately, essentially by considering more colours. The most common format uses eight bits for each of the three RGB components. These images are known as 24-bit true colour and they can contain 16777216 different colours simultaneously. In spite of requiring significantly more memory, the image quality and the continuing reduction in cost of computer memory make this format a good alternative, even for storing the image frames from a video sequence. Of course, a good compression algorithm is always helpful in these cases, particularly, if images need to be transmitted on a network. Here we will consider the processing of grey level images only since they contain enough information to perform feature extraction and image analysis; greater depth on colour analysis/parameterisation is to be found in Chapter 11. Should the image be originally in colour, we will consider processing its luminance only, often computed in a standard way. In any case, the amount of memory used is always related to the image size.



Choosing an appropriate value for the image size,  $N$ , is far more complicated. We want  $N$  to be sufficiently large to resolve the required level of spatial detail in the image. If  $N$  is too small, the image will be coarsely *quantised*: lines will appear to be very ‘blocky’ and some of the detail will be lost. Larger values of  $N$  give more detail, but need more storage space and the images will take longer to process, since there are more pixels. For example, with reference to the image in Figure 2.1(a), Figure 2.2 shows the effect of taking the image at different resolutions. Figure 2.2(a) is a  $64 \times 64$  image, that shows only the broad structure. It is impossible to see any detail in the sitting

subject's face, or anywhere else. Figure 2.2(b) is a  $128 \times 128$  image, which is starting to show more of the detail, but it would be hard to determine the subject's identity. The original image, repeated in Figure 2.2(c), is a  $256 \times 256$  image which shows a much greater level of detail, and the subject can be recognised from the image. Note that the images in Figure 2.2 have been scaled to be the same size. As such, the pixels in Figure 2.2(a) are much larger than in Figure 2.2(c) which emphasises its blocky structure. Common choices are for  $512 \times 512$  or  $1024 \times 1024$  8-bit images which require 256 KB and 1 MB of storage, respectively. If we take a sequence of, say, 20 images for motion analysis, we will need more than 5 MB to store twenty  $512 \times 512$  images. Even though memory continues to become cheaper, this can still impose high cost. But it is not just cost which motivates an investigation of the appropriate image size, the appropriate value for  $N$ . The main question is: are there theoretical guidelines for choosing it? The short answer is 'yes'; the long answer is to look at digital signal processing theory.

The choice of sampling frequency is dictated by the *sampling criterion*. Presenting the sampling criterion requires understanding of how we interpret signals in the *frequency domain*. The way in is to look at the Fourier transform. This is a highly theoretical topic, but do not let that put you off (it leads to image coding, like the JPEG format, so it is very useful indeed). The Fourier transform has found many uses in image processing and understanding; it might appear to be a complex topic (that's actually a horrible pun!) but it is a very rewarding one to study. The particular concern is number of points per unit area or the appropriate sampling frequency of (essentially, the value for  $N$ ), or the rate at which pixel values are taken from, a camera's video signal.

## 2.3 The Fourier Transform

The *Fourier transform* is a way of mapping a signal into its component frequencies. *Frequency* measures in Hertz (Hz) the rate of repetition with time, measured in seconds (s); time is the reciprocal of frequency and vice versa (Hertz = 1/seconds; s = 1/Hz).

Consider a music centre: the sound comes from a computer, a CD player (or a tape, whatever) and is played on the speakers after it has been processed by the amplifier. On the amplifier, you can change the bass or the treble (or the loudness which is a combination of bass and treble). Bass covers the low frequency components and treble covers the high frequency ones. The Fourier transform is a way of mapping the signal, which is a signal varying continuously with time, into its frequency components. When we have transformed the signal, we know which frequencies made up the original sound.

So why do we do this? We have not changed the signal, only its representation. We can now visualise it in terms of its frequencies, rather than as a voltage which changes with time. However, we can now change the frequencies (because we can see them clearly) and this will change the sound. If, say, there is hiss on the original signal then since hiss is a high frequency component, it will show up as a high frequency component in the Fourier transform. So we can see how to remove it by looking at the Fourier transform. If you have ever used a graphic equaliser, you have done this before. The graphic equaliser is a way of changing a signal by interpreting its frequency domain representation; you can selectively control the frequency content by changing the positions of the controls of the graphic equaliser. The equation which defines the *Fourier transform*,  $Fp$ , of a signal  $p$ , is given by a complex integral:

$$Fp(\omega) = \mathfrak{F}(p(t)) = \int_{-\infty}^{\infty} p(t)e^{-j\omega t} dt \quad (2.1)$$

where:  $Fp(\omega)$  is the Fourier transform, and  $\mathfrak{F}$  denotes the Fourier transform process;

$\omega$  is the angular frequency,  $\omega = 2\pi f$  measured in radians/s (where the frequency  $f$  is the reciprocal of time  $t$ ,  $f = 1/t$ );



$j$  is the complex variable  $j = \sqrt{-1}$  (electronic engineers prefer  $j$  to  $i$  since they cannot confuse it with the symbol for current; perhaps they don't want to be mistaken for mathematicians who use  $i = \sqrt{-1}$  )

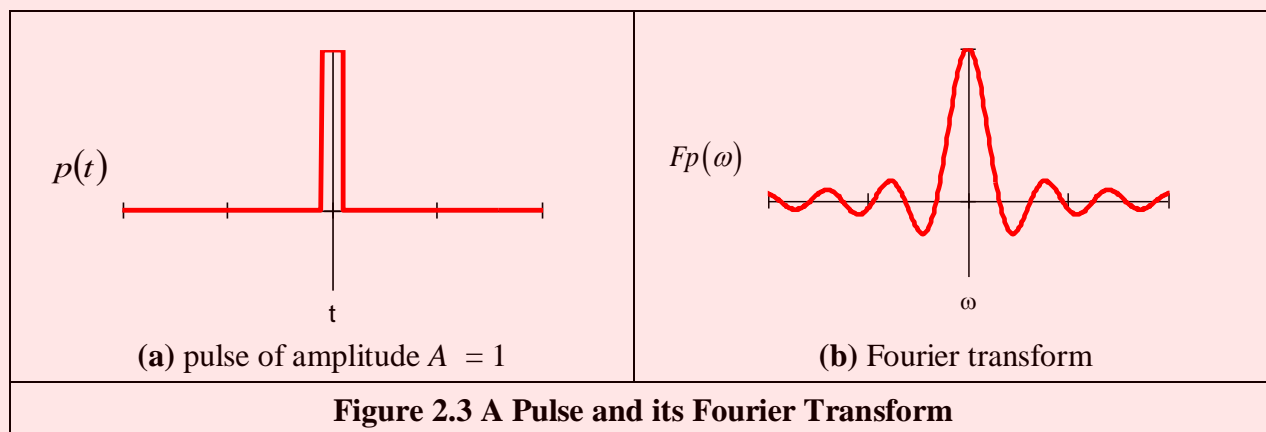
$p(t)$  is a *continuous signal* (varying continuously with time); and

$e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$  gives the frequency components in  $p(t)$ .

We can derive the Fourier transform by applying Equation 2.1 to the signal of interest. We can see how it works by constraining our analysis to simple signals. (We can then say that complicated signals are just made up by adding up lots of simple signals.) If we take a pulse which is of amplitude (size)  $A$  between when it starts at time  $t = -T/2$  and it ends at  $t = T/2$ , and is zero elsewhere, the *pulse* is:

$$p(t) = \begin{cases} A & \text{if } -T/2 \leq t \leq T/2 \\ 0 & \text{otherwise} \end{cases} \quad (2.2)$$

To obtain the Fourier transform, we substitute for  $p(t)$  in Equation 2.1.  $p(t) = A$  only for a specified time so we choose the limits on the integral to be the start and end points of our pulse (it is zero elsewhere) and set  $p(t) = A$ , its value in this time interval. The Fourier transform of this pulse is the result of computing:



$$Fp(\omega) = \int_{-T/2}^{T/2} A e^{-j\omega t} dt \quad (2.3)$$

When we solve this we obtain an expression for  $Fp(\omega)$  :

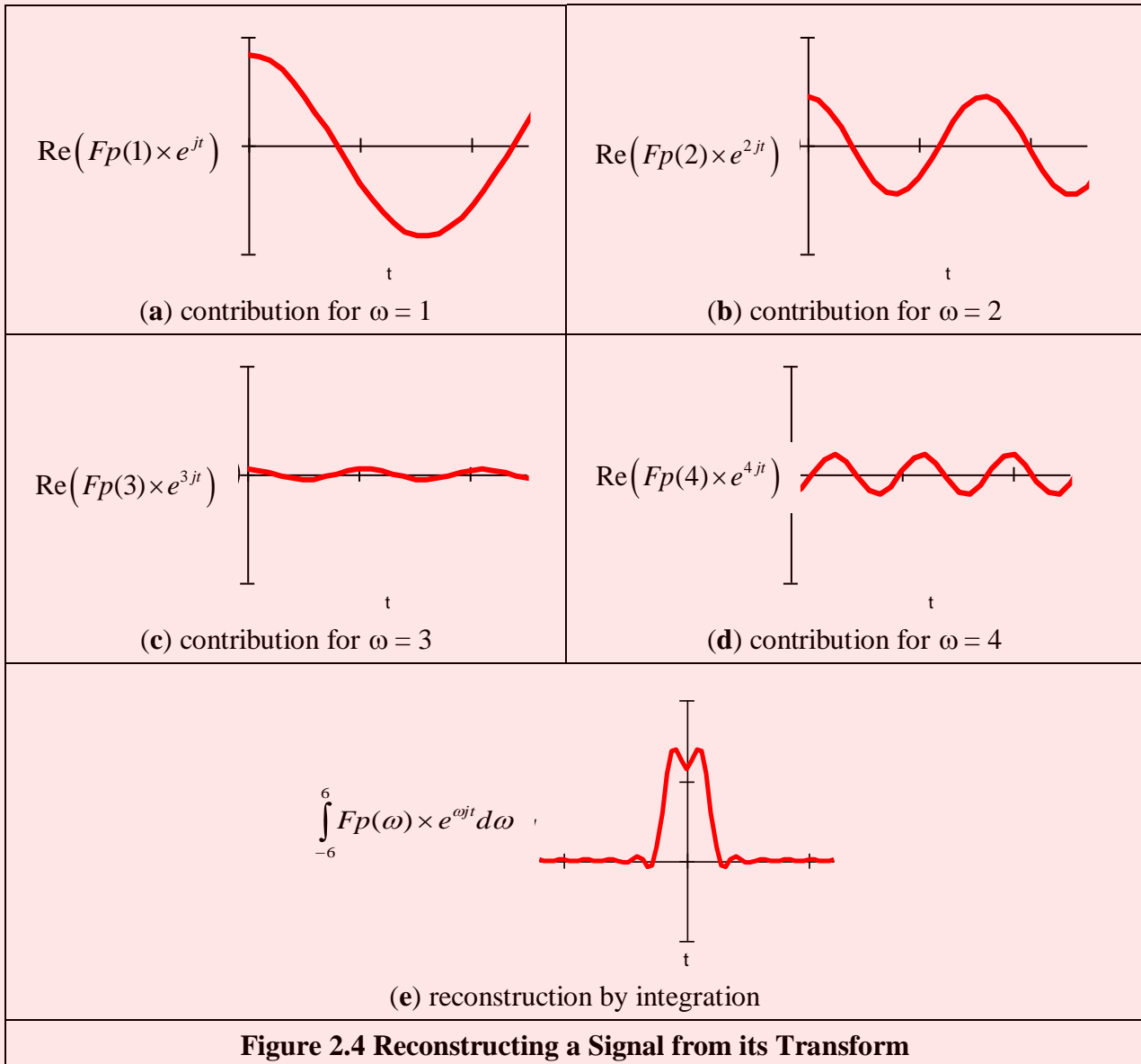
$$Fp(\omega) = -\frac{Ae^{-j\omega T/2} - Ae^{j\omega T/2}}{j\omega} \quad (2.4)$$

By simplification, using the relation  $\sin(\theta) = (e^{j\theta} - e^{-j\theta})/2j$ , then the Fourier transform of the pulse is:

$$Fp(\omega) = \begin{cases} \frac{2A}{\omega} \sin\left(\frac{\omega T}{2}\right) & \text{if } \omega \neq 0 \\ AT & \text{if } \omega = 0 \end{cases} \quad (2.5)$$

This is a version of the *sinc* function,  $\text{sinc}(x) = \sin(x)/x$ . The original pulse, and its transform are illustrated in Figure 2.3. Equation 2.5 (as plotted in Figure 2.3(b)) suggests that a pulse is made up of a lot of low frequencies (the main body of the pulse) and a few higher frequencies (which give us the edges of the pulse). (The range of frequencies is symmetrical around zero frequency; negative

frequency is a necessary mathematical abstraction.) The plot of the Fourier transform is actually called the *spectrum* of the signal, which can be considered akin with the spectrum of light.



**Figure 2.4 Reconstructing a Signal from its Transform**

So what actually is this Fourier transform? It tells us what frequencies make up a time domain signal. The magnitude of the transform at a particular frequency is the amount of that frequency in the original signal. If we collect together sinusoidal signals in amounts specified by the Fourier transform, we should obtain the originally transformed signal. This process is illustrated in Figure 2.4 for the signal and transform illustrated in Figure 2.3. Note that since the Fourier transform is actually a complex number it has real and imaginary parts, and we only plot the real part here. A low frequency, that for  $\omega = 1$ , in Figure 2.4(a) contributes a large component of the original signal; a higher frequency, that for  $\omega = 2$ , contributes less as in Figure 2.4(b). This is because the transform coefficient is less for  $\omega = 2$  than it is for  $\omega = 1$ . There is a very small contribution for  $\omega = 3$ , Figure 2.4(c), though there is more for  $\omega = 4$ , Figure 2.4(d). This is because there are frequencies for which there is no contribution, where the transform is zero. When these signals are integrated together, we achieve a signal that looks similar to our original pulse, Figure 2.4(e). Here we have only considered frequencies from  $\omega = -6$  to  $\omega = 6$ . If the frequency range in integration was larger, more high frequencies would be included, leading to a more faithful reconstruction of the original

pulse.

The result of the Fourier transform is actually a complex number: each value  $Fp(\omega)$  is represented by a pair of numbers that represent its real and imaginary parts. Rather than use the real and imaginary parts, the values are usually represented in terms of *magnitude* (or size, or modulus) and *phase* (or argument). The transform can be represented as:

$$Fp(\omega) = \int_{-\infty}^{\infty} p(t)e^{-j\omega t} dt = \text{Re}(Fp(\omega)) + j \text{Im}(Fp(\omega)) \quad (2.6)$$

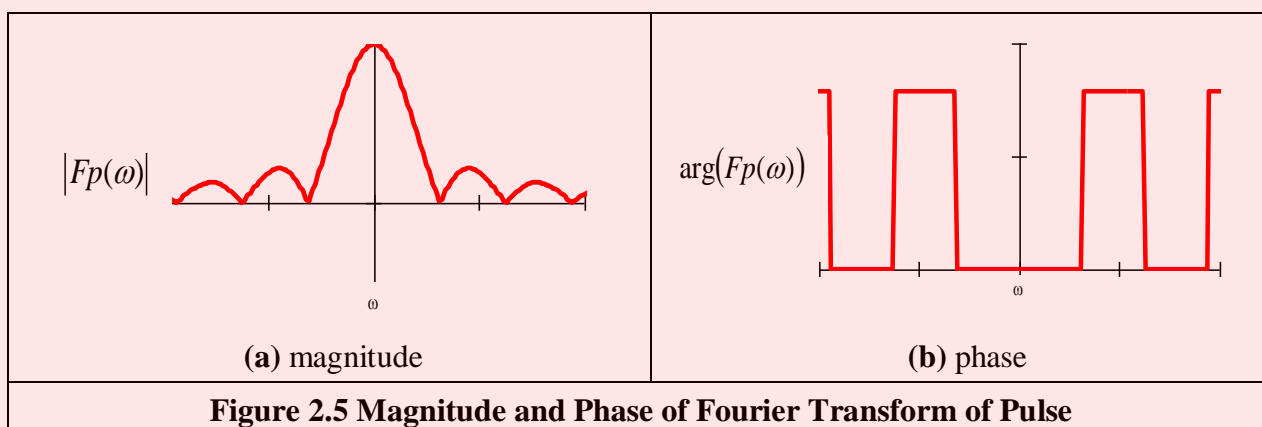
where  $\text{Re}(\ )$  and  $\text{Im}(\ )$  are the real and imaginary parts of the transform, respectively. The magnitude of the transform is then:

$$|Fp(\omega)| = \sqrt{\text{Re}(Fp(\omega))^2 + \text{Im}(Fp(\omega))^2} \quad (2.7)$$

and the phase is:

$$\arg(Fp(\omega)) = \tan^{-1}\left(\frac{\text{Im}(Fp(\omega))}{\text{Re}(Fp(\omega))}\right) \quad (2.8)$$

where the signs of the real and the imaginary components can be used to determine which quadrant the phase (argument) is in, since the phase can vary from 0 to  $2\pi$  radians. The magnitude describes the amount of each frequency component, the phase describes timing, when the frequency components occur. The magnitude and phase of the transform of a pulse are shown in Figure 2.5 where the magnitude returns a positive transform, and the phase is either 0 or  $2\pi$  radians (consistent with the sine function).



In order to return to the time-domain signal, from the frequency domain signal, we require the *inverse Fourier transform*. This was the process illustrated in Figure 2.4, the process by which we reconstructed the pulse from its transform components. The inverse FT calculates  $p(t)$  from  $Fp(\omega)$  by the inverse transformation  $\mathfrak{F}^{-1}$ :

$$p(t) = \mathfrak{F}^{-1}(Fp(\omega)) = \frac{1}{2\pi} \int_{-\infty}^{\infty} Fp(\omega)e^{j\omega t} d\omega \quad (2.9)$$

Together, Equation 2.1 and Equation 2.9 form a relationship known as a *transform pair* that allows us to transform into the frequency domain, and back again. By this process, we can perform operations in the frequency domain or in the time domain, since we have a way of changing between them. One important process is known as *convolution*. The convolution of one signal  $p_1(t)$  with another signal  $p_2(t)$ , where the convolution process denoted by  $*$  is given by the integral

$$p_1(t) * p_2(t) = \int_{-\infty}^{\infty} p_1(\tau) p_2(t - \tau) d\tau \tag{2.10}$$

This is actually the basis of systems theory where the output of a system is the convolution of a stimulus, say  $p_1$ , and a system's response,  $p_2$ . By inverting the time axis of the system response, to give  $p_2(t-\tau)$  we obtain a memory function. The convolution process then sums the effect of a stimulus multiplied by the memory function: the current output of the system is the cumulative response to a stimulus. By taking the Fourier transform of Equation 2.10, the Fourier transform of the convolution of two signals is

$$\begin{aligned} \mathfrak{F}[p_1(t) * p_2(t)] &= \int_{-\infty}^{\infty} \left\{ \int_{-\infty}^{\infty} p_1(\tau) p_2(t - \tau) d\tau \right\} e^{-j\omega t} dt \\ &= \int_{-\infty}^{\infty} \left\{ \int_{-\infty}^{\infty} p_2(t - \tau) e^{-j\omega t} dt \right\} p_1(\tau) d\tau \end{aligned} \tag{2.11}$$

Now since  $\mathfrak{F}[p_2(t - \tau)] = e^{-j\omega\tau} Fp_2(\omega)$  (to be considered later in Section 2.6.1), then

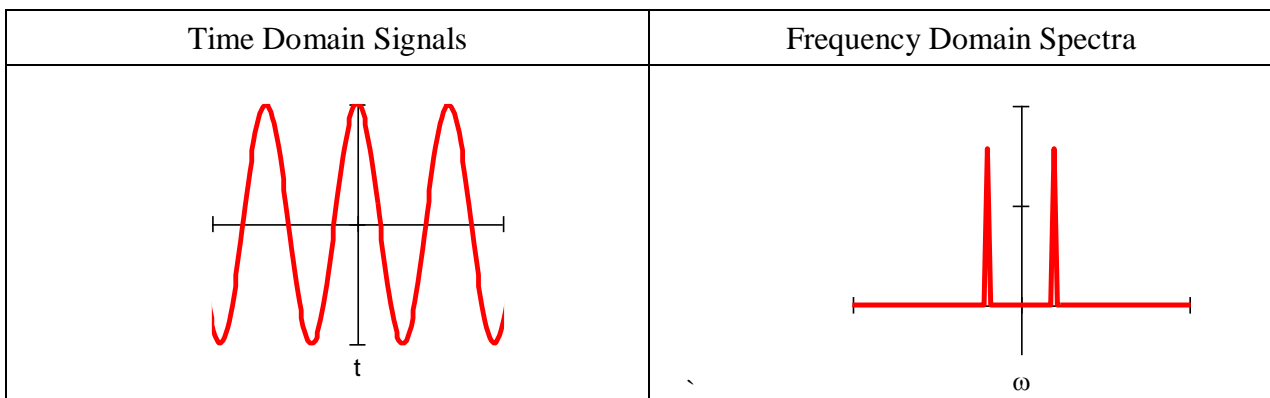
$$\begin{aligned} \mathfrak{F}[p_1(t) * p_2(t)] &= \int_{-\infty}^{\infty} Fp_2(\omega) p_1(\tau) e^{-j\omega\tau} d\tau \\ &= Fp_2(\omega) \int_{-\infty}^{\infty} p_1(\tau) e^{-j\omega\tau} d\tau \\ &= Fp_2(\omega) \times Fp_1(\omega) \end{aligned} \tag{2.12}$$

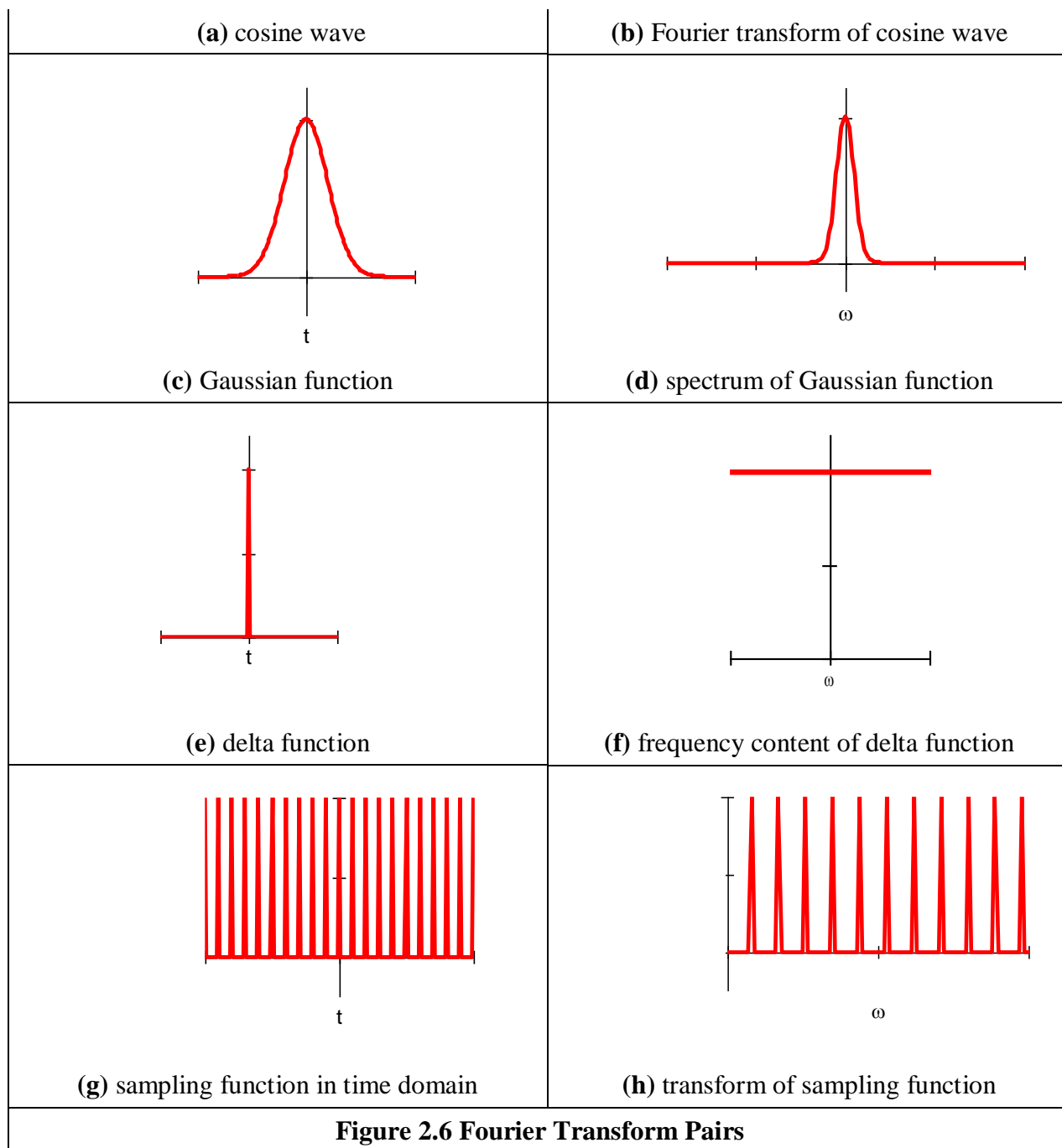
As such, the frequency domain dual of convolution is multiplication; the convolution integral can be performed by *inverse Fourier transformation* of the product of the transforms of the two signals. A frequency domain representation essentially presents signals in a different way but it also provides a different way of processing signals. Later we shall use the duality of convolution to speed up the computation of vision algorithms considerably.

Further, *correlation* is defined to be

$$p_1(t) \otimes p_2(t) = \int_{-\infty}^{\infty} p_1(\tau) p_2(t + \tau) d\tau \tag{2.13}$$

where  $\otimes$  denotes correlation ( $\odot$  is another symbol which is used sometimes, but there is not much consensus on this symbol – if comfort is needed: “in esoteric astrology  $\odot$  represents the creative spark of divine consciousness” no less!). Correlation gives a measure of the match between the two signals  $p_2(\omega)$  and  $p_1(\omega)$ . When  $p_2(\omega) = p_1(\omega)$  we are correlating a signal with itself and the process is known as *autocorrelation*. We shall be using correlation later, to find things in images.





Before proceeding further, we also need to define the *delta function*, which can be considered to be a function occurring at a particular time interval:

$$\text{delta}(t - \tau) = \begin{cases} 1 & \text{if } t = \tau \\ 0 & \text{otherwise} \end{cases} \quad (2.14)$$

The relationship between a signal's time domain representation and its frequency domain version is also known as a *transform pair*: the transform of a pulse (in the time domain) is a sinc function in the frequency domain. Since the transform is symmetrical, the Fourier transform of a sinc function is a pulse.

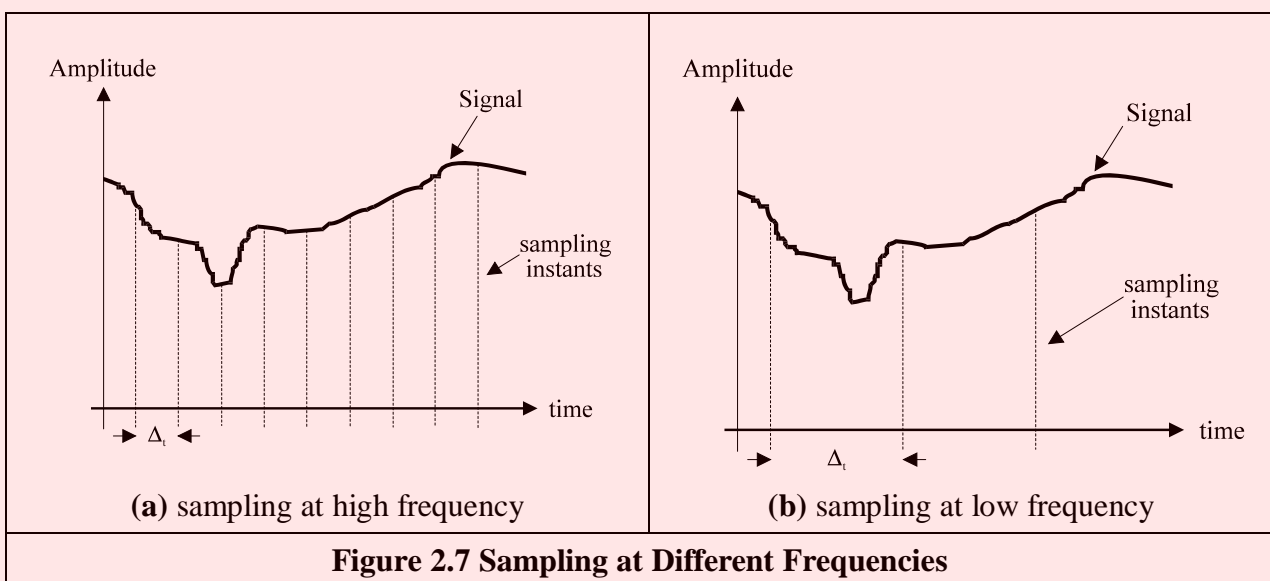
There are other Fourier transform pairs, as illustrated in Figure 2.6. Firstly, Figures 2.6(a) and (b) show that the Fourier transform of a cosine function is two points in the frequency domain (at

the same value for positive and negative frequency) – we expect this since there is only one frequency in the cosine function, the frequency shown by its transform. Figures 2.6(c) and (d) show that the transform of the *Gaussian function* is another Gaussian function, this illustrates linearity (for linear systems it's Gaussian in, Gaussian out which is another version of GIGO). Figure 2.6(e) is a single point (the delta function) which has a transform that is an infinite set of frequencies, Figure 2.6(f), an alternative interpretation is that a delta function contains an equal amount of all frequencies. This can be explained by using Equation 2.5 where if the pulse is of shorter duration ( $T$  tends to zero), the sinc function is wider; as the pulse becomes infinitely thin, the spectrum becomes infinitely flat.

Finally, Figures 2.6(g) and (h) show that the transform of a set of uniformly-spaced delta functions is another set of uniformly-spaced delta functions, but with a different spacing. The spacing in the frequency domain is the reciprocal of the spacing in the time domain. By way of a (non-mathematical) explanation, let us consider that the Gaussian function in Figure 2.6(c) is actually made up by summing a set of closely spaced (and very thin) Gaussian functions. Then, since the spectrum for a delta function is infinite, as the Gaussian function is stretched in the time domain (eventually to be a set of pulses of uniform height) we obtain a set of pulses in the frequency domain, but spaced by the reciprocal of the time domain spacing. This transform pair is actually the basis of sampling theory (which we aim to use to find a criterion which guides us to an appropriate choice for the image size).

## 2.4 The Sampling Criterion

The *sampling criterion* specifies the condition for the correct choice of sampling frequency. *Sampling* concerns taking instantaneous values of a continuous signal, physically these are the outputs of an A/D converter sampling a camera signal. Clearly, the samples are the values of the signal at sampling instants. This is illustrated in Figure 2.7 where Figure 2.7(a) concerns taking samples at a high frequency (the spacing between samples is low), compared with the amount of change seen in the signal of which the samples are taken. Here, the samples are taken sufficiently fast to notice the slight dip in the sampled signal. Figure 2.7(b) concerns taking samples at a low frequency, compared with the rate of change of (the maximum frequency in) the sampled signal. Here, the slight dip in the sampled signal is not seen in the samples taken from it.



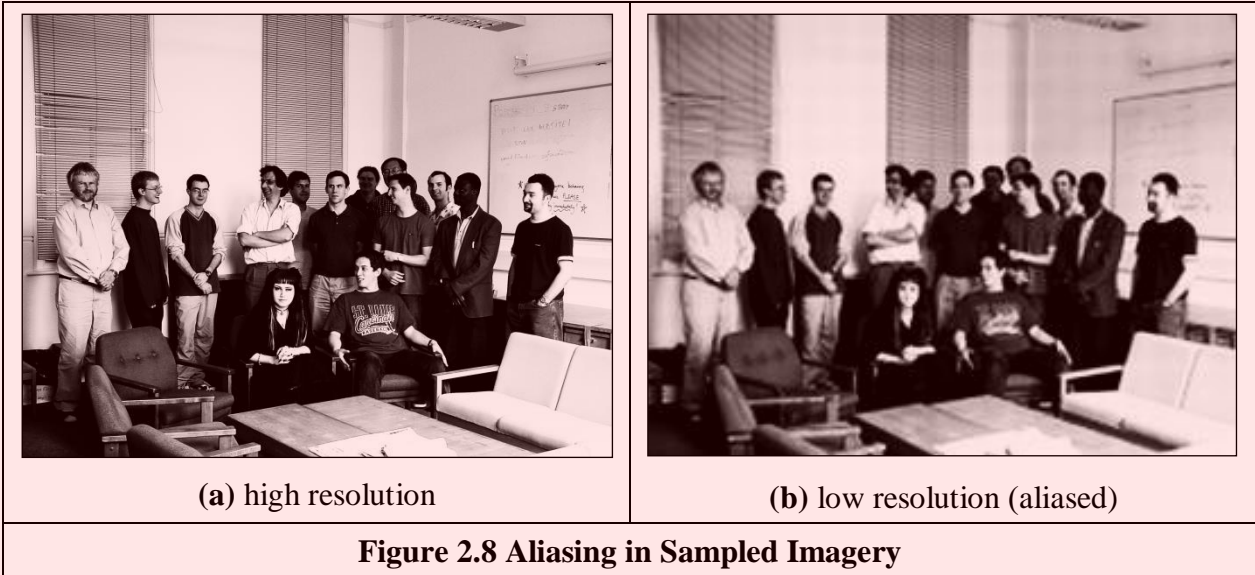
We can understand the process better in the frequency domain. Let us consider a time-variant signal which has a range of frequencies between  $-f_{max}$  and  $f_{max}$  as illustrated in Figure 2.9(b). This range of frequencies is shown by the Fourier transform where the signal's spectrum exists only between these frequencies. This function is sampled every  $\Delta_t$  s: this is a sampling function of spikes occurring every  $\Delta_t$  s. The Fourier transform of the sampling function is a series of spikes separated by  $f_{sample} = 1/\Delta_t$  Hz. The Fourier pair of this transform was illustrated earlier, Figures 2.6(g) and (h).

The sampled signal is the result of multiplying the time-variant signal by the sequence of spikes, this gives samples that occur every  $\Delta_t$  s, and the sampled signal is shown in Figure 2.9(a). These are the outputs of the A/D converter at sampling instants. The frequency domain analogue of this sampling process is to convolve the spectrum of the time-variant signal with the spectrum of the sampling function. Convolution of the signals, the convolution process, implies that we take the spectrum of one, flip it along the horizontal axis and then slide it across the other. Taking the spectrum of the time-variant signal and sliding it over the spectrum of the spikes, results in a spectrum where the spectrum of the original signal is repeated every  $1/\Delta_t$  Hz,  $f_{sample}$  in Figure 2.9(b-d). If the spacing between samples is  $\Delta_t$ , the repetitions of the time-variant signal's spectrum are spaced at intervals of  $1/\Delta_t$ , as in Figure 2.9(b). If the sample spacing is large, then the time-variant signal's spectrum is replicated close together and the spectra collide, or interfere, as in Figure 2.9(d). The spectra just touch when the sampling frequency is twice the maximum frequency in the signal. If the frequency domain spacing,  $f_{sample}$ , is more than twice the maximum frequency,  $f_{max}$ , the spectra do not collide or interfere, as in Figure 2.9(c). If the sampling frequency exceeds twice the maximum frequency then the spectra cannot collide. This is the *Nyquist sampling criterion*:

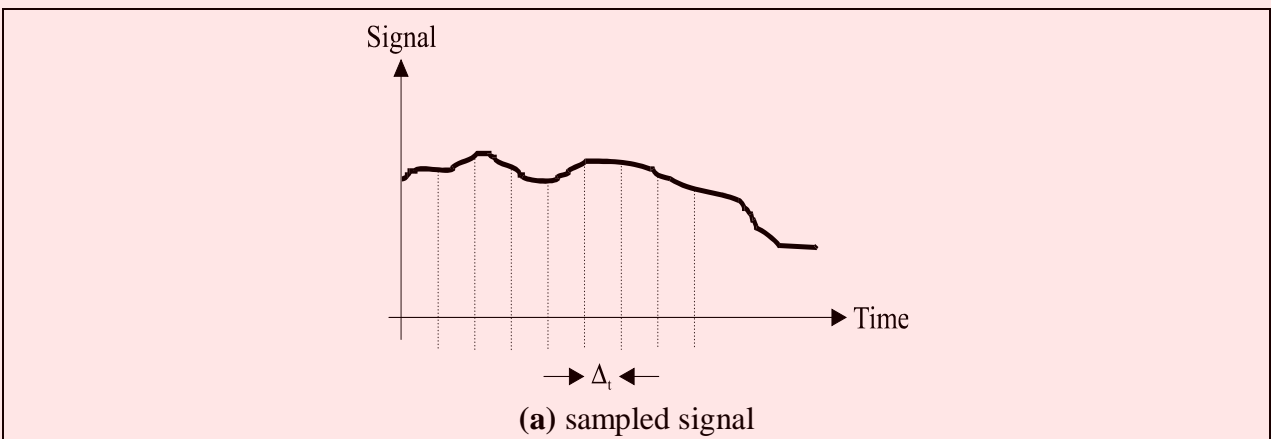
In order to reconstruct a signal from its samples, the sampling frequency must be at least twice the highest frequency of the sampled signal.

If we do not obey Nyquist's sampling theorem the spectra collide. When we inspect the sampled signal, whose spectrum is within  $-f_{max}$  to  $f_{max}$ , wherein the spectra collided, the corrupt spectrum implies that by virtue of sampling, we have ruined some of the information. If we were to attempt to reconstruct a signal by inverse Fourier transformation of the sampled signal's spectrum, processing Figure 2.9(d) would lead to the wrong signal whereas inverse Fourier transformation of the frequencies between  $-f_{max}$  and  $f_{max}$  in Figures 2.9(b) and (c) would lead back to the original signal. This can be seen in computer images as illustrated in Figure 2.8 which show an image of a group of people (the computer vision research team at Southampton) displayed at different spatial resolutions (the contrast has been increased to the same level in each sub-image, so that the effect we want to demonstrate should definitely show up in the print copy). Essentially, the people become less distinct in the lower resolution image, Figure 2.8(b). Now, look closely at the window blinds behind the people. At higher resolution, in Figure 2.8(a), these appear as normal window blinds. In Figure 2.8(b), which is sampled at a much lower resolution, a new pattern appears: the pattern appears to be curved - and if you consider the blinds' relative size the shapes actually appear to be much larger than normal window blinds. So by reducing the resolution, we are seeing something different, an *alias* of the true information - something that is not actually there at all, but appears to be there by result of sampling. This is the result of sampling at too low a frequency: if we sample at high frequency, the interpolated result matches the original signal; if we sample at too low a frequency we can get the wrong signal. (For these reasons people on television tend to wear non-

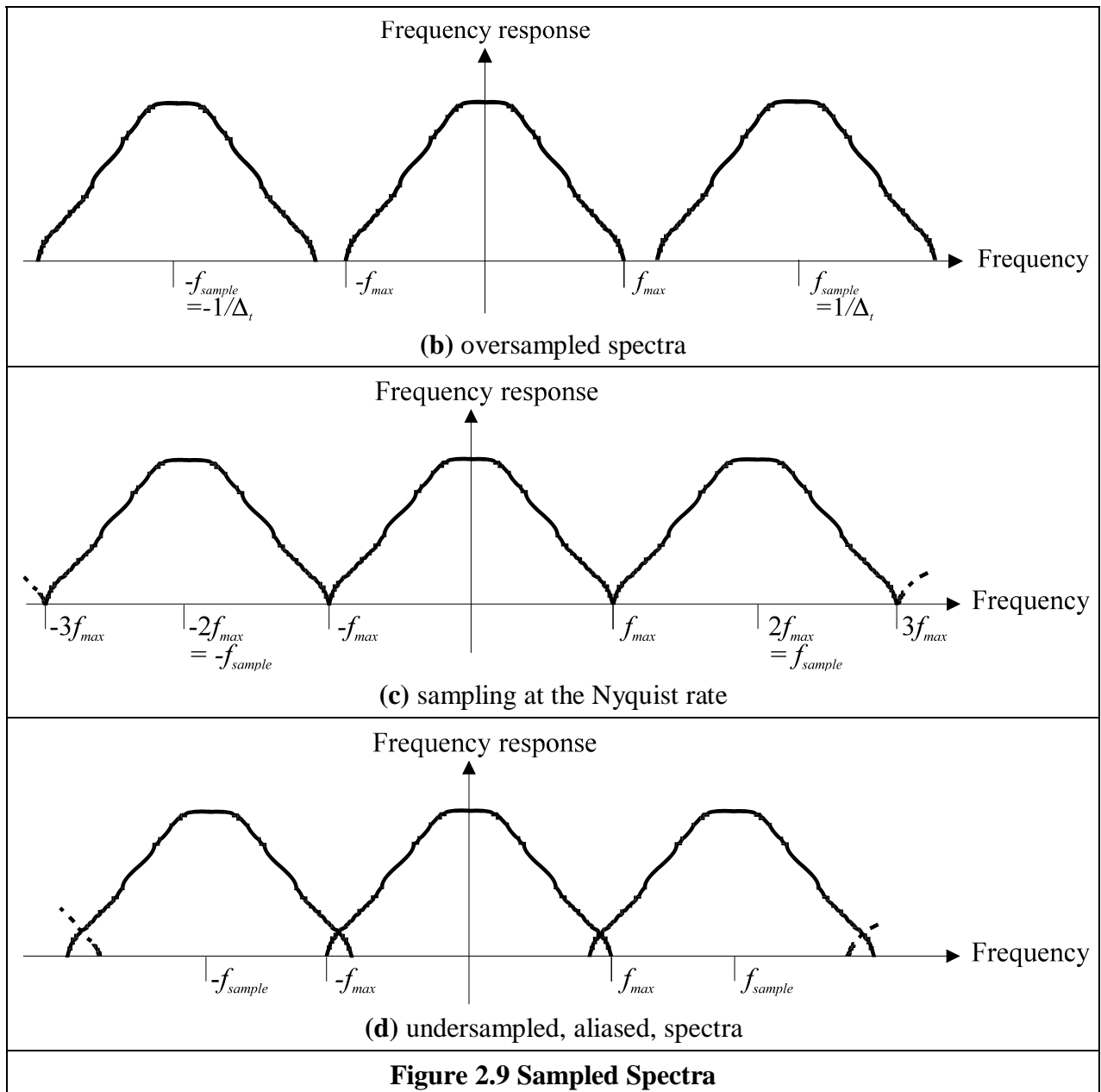
chequered clothes – or should not!). Note that this effect can be seen, in the way described, in the printed version of this book. This is because the printing technology is very high resolution. If you were to print this page offline (and sometimes even to view it), e.g. from a Google Books sample, the nature of the effect of aliasing depends on the resolution of the printed image, and so the aliasing effect might also be seen in the high resolution image as well as in the low resolution version – which rather spoils the point.



In art, Dali’s picture “Gala Contemplating the Mediterranean Sea, which at 20 meters becomes the portrait of Abraham Lincoln” (Homage to Rothko) is a classic illustration of sampling. At high resolution you see a detailed surrealist image, with Mrs Dali as a central figure. Viewed from a distance - or for the shortsighted, without your spectacles on - the image becomes a (low resolution) picture of Abraham Lincoln. For a more modern view of sampling [Unser00] is well worth a look. The *compressive sensing* approach [Donoho06] takes advantage of the fact that many signals have components that are significant, or nearly zero, leading to cameras which acquire significantly fewer elements to represent an image. This provides an alternative basis for compressed image acquisition without loss of resolution.



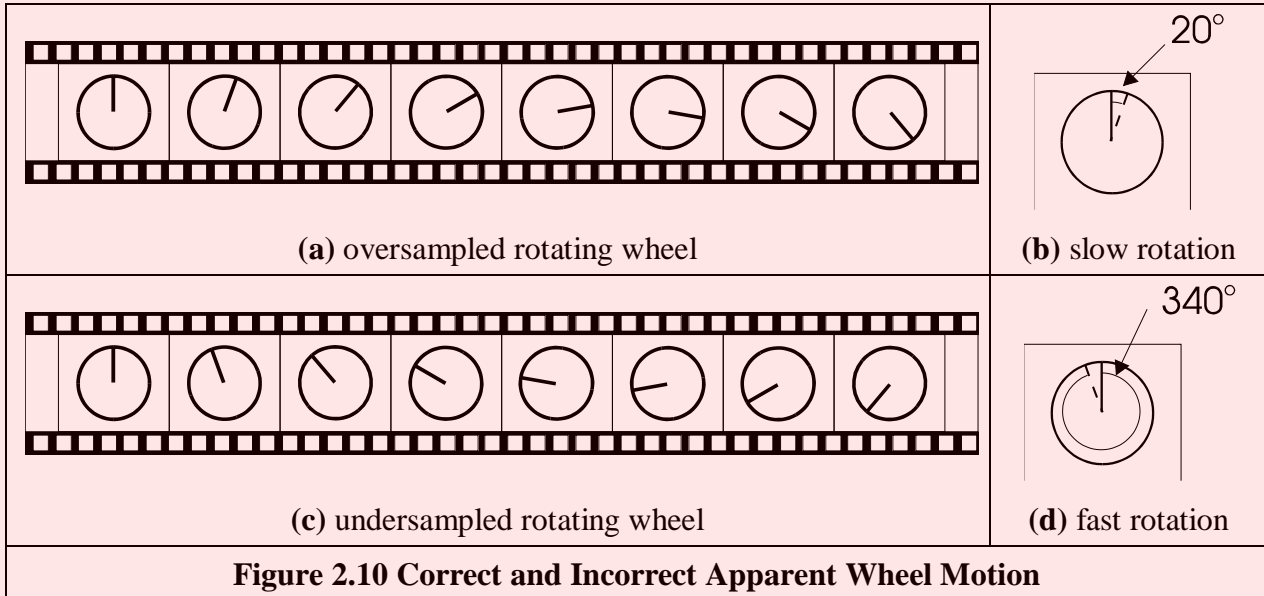




Obtaining the wrong signal is called *aliasing*: our interpolated signal is an alias of its proper form. Clearly, we want to avoid aliasing, so according to the sampling theorem we must sample at twice the maximum frequency of the signal coming out of the camera. The maximum frequency is defined to be 5.5 MHz so we must sample the camera signal at 11 MHz. (For information, when using a computer to analyse speech we must sample the speech at a minimum frequency of 12 kHz since the maximum speech frequency is 6 kHz.) Given the timing of a video signal, sampling at 11 MHz implies a minimum image resolution of  $576 \times 576$  pixels. This is unfortunate: 576 is not an integer power of two which has poor implications for storage and processing. Accordingly, since some image processing systems have a maximum resolution of  $512 \times 512$ , they must anticipate aliasing. This is mitigated somewhat by the observations that:

1. globally, the lower frequencies carry more information whereas locally the higher frequencies contain more information so the corruption of high frequency information is of less importance; and
2. there is limited depth of focus in imaging systems (reducing high frequency content).

But aliasing can, and does, occur and we must remember this when interpreting images. A different form of this argument applies to the images derived from digital cameras. The basic argument that the precision of the estimates of the high order frequency components is dictated by the relationship between the effective sampling frequency (the number of image points) and the imaged structure, naturally still applies.



The effects of sampling can often be seen in films, especially in the rotating wheels of cars, as illustrated in Figure 2.10. This shows a wheel with a single spoke, for simplicity. The film is a sequence of frames starting on the left. The sequence of frames plotted Figure 2.10(a) is for a wheel which rotates by  $20^\circ$  between frames, as illustrated in Figure 2.10(b). If the wheel is rotating much faster, by  $340^\circ$  between frames, as in Figure 2.10(c) and Figure 2.10(d), to a human viewer the wheel will appear to rotate in the opposite direction. If the wheel rotates by  $360^\circ$  between frames, it will appear to be stationary. In order to perceive the wheel as rotating forwards, then the rotation between frames must be  $180^\circ$  at most. This is consistent with sampling at more than twice the maximum frequency. Our eye can resolve this in films (when watching a film, we bet you haven't thrown a wobbly because the car's going forwards whereas the wheels say it's going the other way) since we know that the direction of the car must be consistent with the motion of its wheels, and we expect to see the wheels appear to go the wrong way, sometimes.

## 2.5 The Discrete Fourier Transform (DFT)

### 2.5.1 One Dimensional Transform

Given that image processing concerns sampled data, we require a version of the Fourier transform which handles this. This is known as the *discrete Fourier transform* (DFT). The DFT of a set of  $N$  points  $\mathbf{p}_x$  (sampled at a frequency which at least equals the Nyquist sampling rate) into sampled frequencies  $\mathbf{Fp}_u$  is:

$$\mathbf{Fp}_u = \frac{1}{N} \sum_{x=0}^{N-1} \mathbf{p}_x e^{-j\left(\frac{2\pi}{N}\right)xu} \quad (2.15)$$

where the scaling coefficient  $1/N$  ensures the d.c. coefficient  $\mathbf{Fp}_0$  is the average of all samples. Equation 2.15 is a discrete analogue of the continuous Fourier transform: the continuous signal is

replaced by a set of samples, the continuous frequencies by sampled ones, and the integral is replaced by a summation. If the DFT is applied to samples of a pulse in a window from sample 0 to sample  $N/2 - 1$  (when the pulse ceases), the equation becomes:

$$\mathbf{Fp}_u = \frac{1}{N} \sum_{x=0}^{\frac{N}{2}-1} A e^{-j\left(\frac{2\pi}{N}\right)xu} \quad (2.16)$$

And since the sum of a geometric progression can be evaluated according to:

$$\sum_{k=0}^n a_0 r^k = \frac{a_0 (1 - r^{n+1})}{1 - r} \quad (2.17)$$

the discrete Fourier transform of a sampled pulse is given by:

$$\mathbf{Fp}_u = \frac{A}{N} \left( \frac{1 - e^{-j\left(\frac{2\pi}{N}\right)\left(\frac{N}{2}\right)u}}{1 - e^{-j\left(\frac{2\pi}{N}\right)u}} \right) \quad (2.18)$$

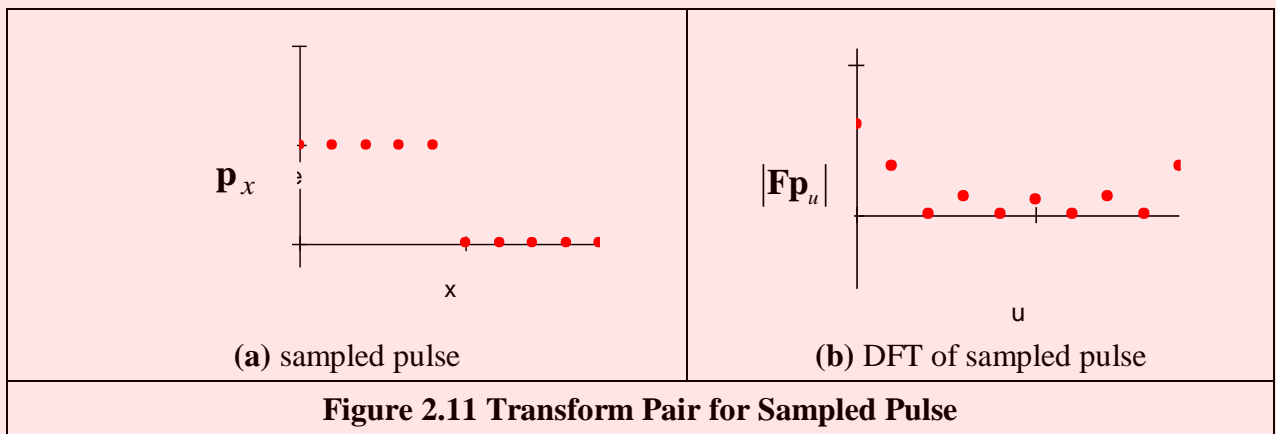
By rearrangement, we obtain:

$$\mathbf{Fp}_u = \frac{A}{N} e^{-j\left(\frac{\pi u}{2}\right)\left(1 - \frac{2}{N}\right)} \frac{\sin(\pi u / 2)}{\sin(\pi u / N)} \quad (2.19)$$

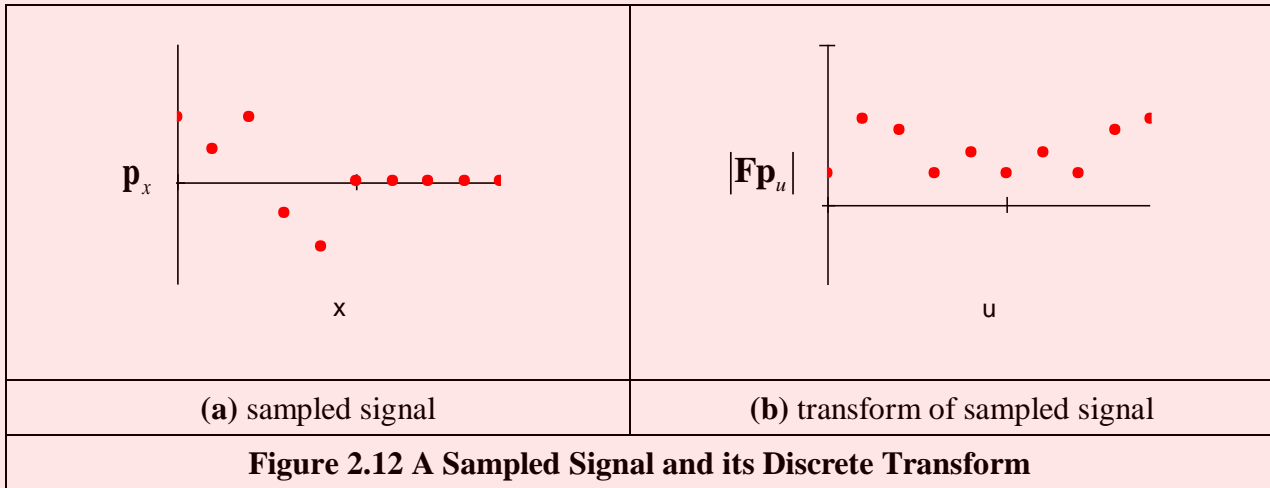
The modulus of the transform is:

$$|\mathbf{Fp}_u| = \frac{A}{N} \left| \frac{\sin(\pi u / 2)}{\sin(\pi u / N)} \right| \quad (2.20)$$

since the magnitude of the exponential function is 1. The original pulse is plotted Figure 2.11(a) and the magnitude of the Fourier transform plotted against frequency is given in Figure 2.11(b)



This is clearly comparable with the result of the continuous Fourier transform of a pulse, Figure 2.3, since the transform involves a similar, sinusoidal, signal. The spectrum is equivalent to a set of sampled frequencies; we can build up the sampled pulse by adding up the frequencies according to the Fourier description. Consider a signal such as that shown in Figure 2.12(a). This has no explicit analytic definition, as such it does not have a closed Fourier transform; the Fourier transform is generated by direct application of Equation 2.15. The result is a set of samples of frequency, Figure 2.12(b).

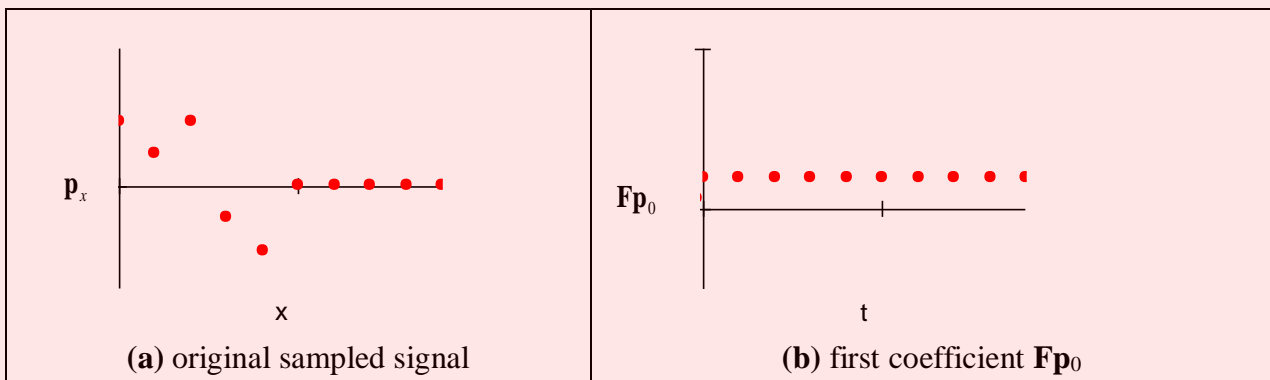


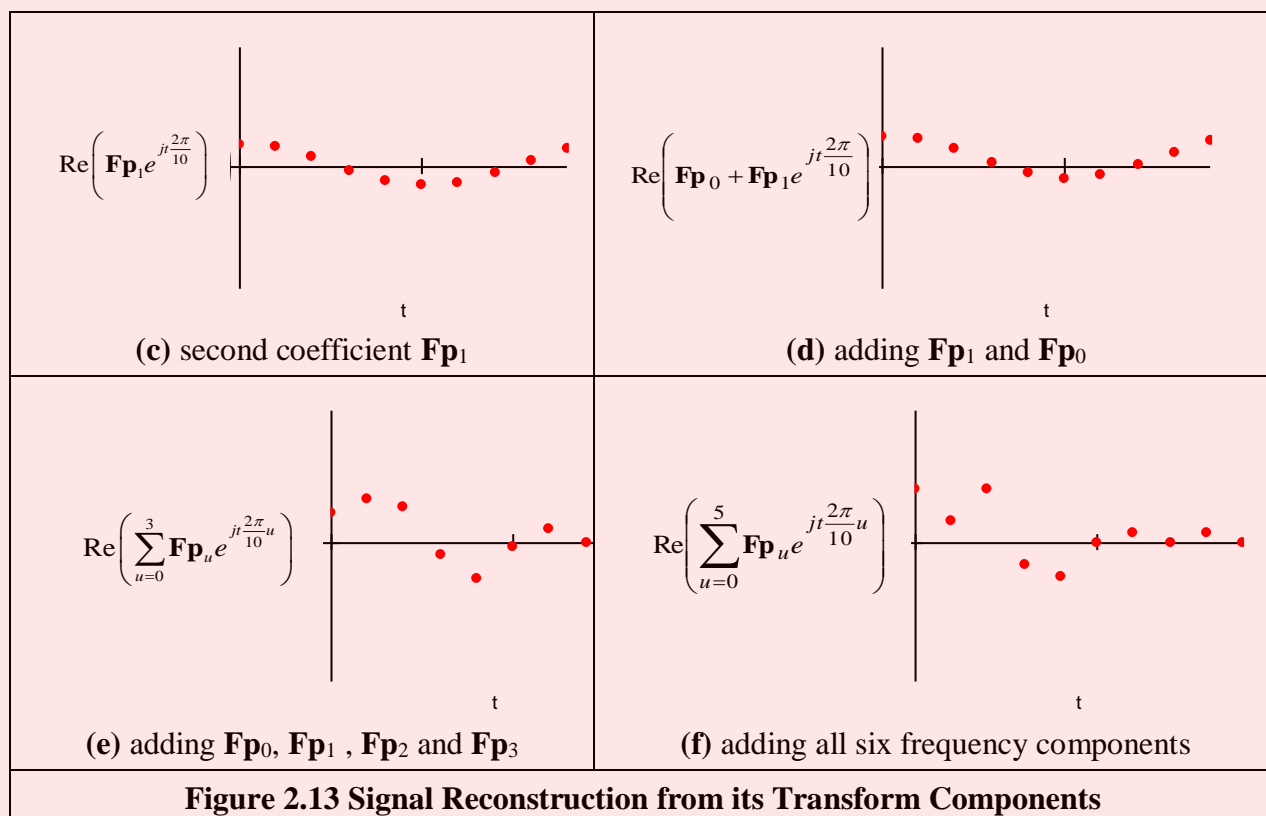
The Fourier transform in Figure 2.12(b) can be used to reconstruct the original signal in Figure 2.12(a), as illustrated in Figure 2.13. Essentially, the coefficients of the Fourier transform tell us how much there is of each of a set of sinewaves (at different frequencies), in the original signal. The lowest frequency component  $Fp_0$ , for zero frequency, is called the *d.c. component* (it is constant and equivalent to a sinewave with no frequency) and it represents the average value of the samples. Adding the contribution of the first coefficient  $Fp_0$ , Figure 2.13(b), to the contribution of the second coefficient  $Fp_1$ , Figure 2.13(c), is shown in Figure 2.13(d). This shows how addition of the first two frequency components approaches the original sampled signal. The approximation improves when the contribution due to the fourth component,  $Fp_3$ , is included, as shown in Figure 2.13(e). Finally, adding up all six frequency components gives a close approximation to the original signal, as shown in Figure 2.13(f).

This process is, of course, the *inverse DFT*. This can be used to reconstruct a sampled signal from its frequency components by:

$$p_x = \sum_{u=0}^{N-1} Fp_u e^{j\left(\frac{2\pi}{N}\right)ux} \tag{2.21}$$

Note that there are several assumptions made prior to application of the DFT. The first is that the sampling criterion has been satisfied. The second is that the sampled function replicates to infinity. When generating the transform of a pulse, Fourier theory assumes that the pulse repeats outside the window of interest. (There are window operators that are designed specifically to handle difficulty at the ends of the sampling window.) Finally, the maximum frequency corresponds to half the sampling period. This is consistent with the assumption that the sampling criterion has not been violated, otherwise the high frequency spectral estimates will be corrupted.



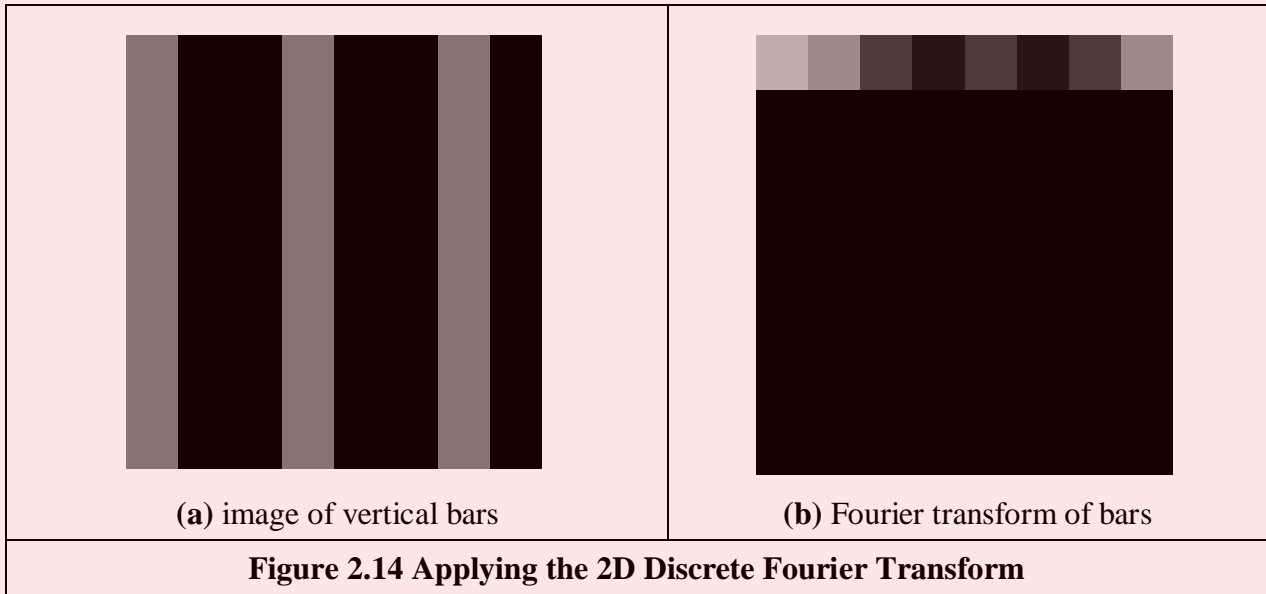


## 2.5.2 Two Dimensional Transform

Equation 2.15 gives the DFT of a one-dimensional signal. We need to generate Fourier transforms of images so we need a *two-dimensional discrete Fourier transform*. This is a transform of pixels (sampled picture points) with a two dimensional spatial location indexed by coordinates  $x$  and  $y$ . This implies that we have two dimensions of frequency,  $u$  and  $v$ , which are the horizontal and vertical spatial frequencies, respectively. Given an image of a set of vertical lines, the Fourier transform will show only horizontal spatial frequency. The vertical spatial frequencies are zero since there is no vertical variation along the  $y$  axis. The two dimensional Fourier transform evaluates the frequency data,  $\mathbf{FP}_{u,v}$ , from the  $N \times N$  pixels  $\mathbf{P}_{x,y}$  as:

$$\mathbf{FP}_{u,v} = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j \left( \frac{2\pi}{N} \right) (ux+vy)} \quad (2.22)$$

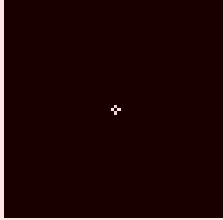
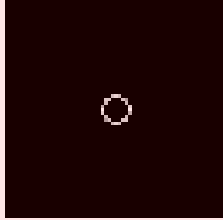
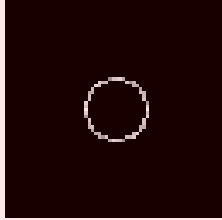
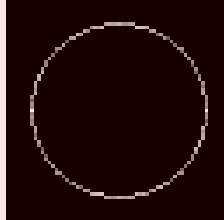
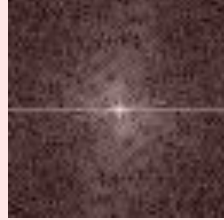
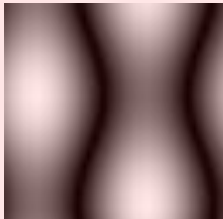
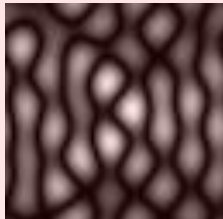
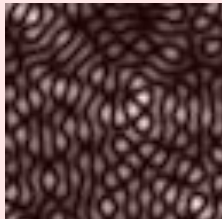

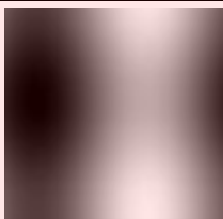
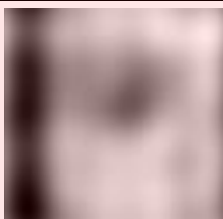

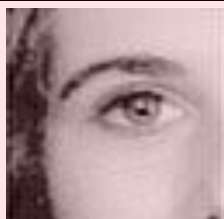
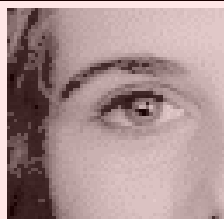
Where the scaling coefficient  $1/N^2$  makes the d.c. coefficient  $\mathbf{FP}_{0,0}$  equal the average of all points in the image (in Matlab the scaling coefficient is 1.0). The Fourier transform of an image can actually be obtained *optically* by transmitting a laser through a photographic slide and forming an image using a lens. The Fourier transform of the image of the slide is formed in the front focal plane of the lens. This is still restricted to transmissive systems whereas reflective formation would widen its application potential considerably (since optical computation is just slightly faster than its digital counterpart). The magnitude of the 2D DFT of an image of vertical bars (Figure 2.14(a)) is shown in Figure 2.14(b). This shows that there are only horizontal spatial frequencies; the image is constant in the vertical axis and there are no vertical spatial frequencies.



The *two-dimensional (2D) inverse DFT* transforms from the frequency domain back to the image domain, to reconstruct the image. The 2D inverse DFT is given by:

$$\mathbf{P}_{x,y} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \mathbf{FP}_{u,v} e^{j\left(\frac{2\pi}{N}\right)(ux+vy)} \quad (2.23)$$

The contribution of different frequencies is illustrated in Figure 2.15 where we have images showing in the first row (a)-(d) the position of the image transform components (presented as  $\log[\text{magnitude}]$ ), in the second row (f)-(i) the image constructed from that single component, and in the third row (j)-(m) the reconstruction (by the inverse FT) using frequencies up to and including that component. There is also the image of the magnitude of the Fourier transform, (e). We shall take the transform components from a circle centred at the middle of the transform image. In Figure 2.15 the first column is the transform components at radius 1 (which are low frequency components), the second column is the radius 4 components, the third column is at radius 9 and the fourth column is the radius 25 components (the higher frequency components). The last column has the complete Fourier transform image, (e), and the reconstruction of the image from the transform, (n). As we include more components we include more detail; the lower order components carry the bulk of the shape, not the detail. In the bottom row, the first components plus the d.c. component give a very coarse approximation Figure 2.15(j) when the components up to radius four are added we can see the shape of a face Figure 2.15(k); the components up to radius 9 order allow us to see the face features Figure 2.15(l), but they are not sharp; we can infer identity from the components up to radius 25 Figure 2.15(m), noting that there are still some image artefacts on the right hand side of the image; when all components are added Figure 2.15(n) we return to the original image. This also illustrates *coding*, as the image can be encoded by retaining fewer of the components of the image than are in the complete transform. Figure 2.15 (m) is a good example of where an image of acceptable quality can be reconstructed, even when about half of the components are discarded. There are considerably better coding approaches than this, though we shall not consider coding in this text, and compression ratios can be considerably higher and still achieve acceptable quality. Note that it is common to use logarithms of magnitude to display Fourier transforms (Section 3.3.1) as otherwise the magnitude of the d.c. component can make the transform difficult to see.

				
(a) transform radius 1 components	(b) transform radius 4 components	(c) transform radius 9 components	(d) transform radius 25 components	(e) complete transform
				
(f) image by radius 1 components	(g) image by radius 4 components	(h) image by radius 9 components	(i) image by radius 25 components	
				
(j) reconstruction up to 1 <sup>st</sup>	(k) reconstruction up to 4 <sup>th</sup>	(l) reconstruction up to 9 <sup>th</sup>	(m) reconstruction up to 25 <sup>th</sup>	(n) reconstruction with all
<b>Figure 2.15 Image Reconstruction and Different Frequency Components</b>				

One of the important properties of the FT is *replication* which implies that the transform repeats in frequency up to infinity, as indicated in Figure 2.9 for 1D signals. To show this for 2D signals, we need to investigate the Fourier transform, originally given by  $\mathbf{FP}_{u,v}$ , at integer multiples of the number of sampled points  $\mathbf{FP}_{u+mN,v+nN}$  (where  $m$  and  $n$  are integers). The Fourier transform  $\mathbf{FP}_{u+mN,v+nN}$  is, by substitution in Equation 2.22:

$$\mathbf{FP}_{u+mN,v+nN} = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)((u+mN)x+(v+nN)y)} \quad (2.24)$$

so,

$$\mathbf{FP}_{u+mN,v+nN} = \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)} \times e^{-j2\pi(mx+ny)} \quad (2.25)$$

and since  $e^{-j2\pi(mx+ny)} = 1$  (since the term in brackets is always an integer and then the exponent is always an integer multiple of  $2\pi$ ) then

$$\mathbf{FP}_{u+mN,v+nN} = \mathbf{FP}_{u,v} \quad (2.26)$$

which shows that the replication property does hold for the Fourier transform. However, Equation 2.22 and Equation 2.23 are very slow for large image sizes. They are usually implemented by using the *Fast Fourier Transform* (FFT) which is a splendid rearrangement of the Fourier transform's

computation which improves speed dramatically. The FFT algorithm is beyond the scope of this text but is also a rewarding topic of study (particularly for computer scientists or software engineers). The FFT can only be applied to square images whose size is an integer power of 2 (without special arrangement). Calculation actually involves the *separability* property of the Fourier transform. Separability means that the Fourier transform is calculated in two stages: the rows are first transformed using a 1D FFT, then this data is transformed in columns, again using a 1D FFT. This process can be achieved since the sinusoidal *basis functions* are orthogonal. Analytically, this implies that the 2D DFT can be decomposed as in Equation 2.27

$$\mathbf{FP}_{u,v} = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \mathbf{P}_{x,y} e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)} = \frac{1}{MN} \sum_{x=0}^{N-1} \left\{ \sum_{y=0}^{M-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)(vy)} \right\} e^{-j\left(\frac{2\pi}{M}\right)(ux)} \quad (2.27)$$

where  $M$  and  $N$  are the numbers of columns and rows, respectively. Equation 2.27 shows how separability is achieved, since the inner term expresses transformation along one axis (the  $y$  axis), and the outer term transforms this along the other (the  $x$  axis).

```
function [Fourier] = F_transform(image)
image=double(image);
[rows, cols] = size(image);
%we deploy equation 2.27, so that we can handle non square images
for u=1:cols % along the horizontal axis
    for v=1:rows % down the vertical axis
        sumx=0;
        for x=1:cols
            %first we transform the rows
            sumy=0;
            for y=1:rows %Eq 2.27 inner bracket
                sumy=sumy+image(y,x)*exp(-1j*2*pi*(v-1)*(y-1)/rows);
            end
            %then we do the columns Eq 2.27 outer
            sumx=sumx+sumy*exp(-1j*2*pi*(u-1)*(x-1)/cols);
        end %and finally normalise
        Fourier(v,u) = sumx/(rows*cols);
    end
end
```

**Code 2.1 2D DFT, Exponential Form implementing Equation 2.27**

Code 2.1 illustrates the implementation of Equation 2.27 in Matlab. The implementation simply evaluates the complex exponent in the definition. Since the computational cost of a 1D FFT of  $N$  points is  $O(N\log_2(N))$ , the cost (by separability) for the 2D FFT is  $O(N^2\log_2(N))$  whereas the computational cost of the 2D DFT is  $O(N^3)$ . This implies a considerable saving since it suggests that the FFT requires much less time, particularly for large image sizes (so for a  $1024 \times 1024$  image, if the FFT takes seconds, the DFT will take minutes). The 2D FFT is available in Matlab using the `fft2` function which gives a result equivalent to Equations 2.22 or 2.27. You can note the difference in time between executing the code in Code 2.1 (or our Python version) and Matlab's own FFT operator (note there is some difference due to compiled vs interpreted code; do not run the basic version on a large image as it will take a very long time). The inverse 2D FFT, Equation 2.23, can be implemented using the Matlab `ifft2` function. (The difference between many Fourier transform implementations essentially concerns the chosen scaling factor, though the order of the frequency components differs from the basic equations in the Matlab functions.) The direct Matlab implementation of the 2D DFT in Equation 2.27 is given in Code 2.1. This is simply called using



the command `b=F_Transform(a)` and the routine enforces the change from an integer format to double precision, as needed when using complex numbers in Matlab. It is easier to work in double precision throughout when developing code; integer formats can be used to speed real implementations (with caution: an early ARIANE space rocket blew up given erroneous conversion of a 32-bit integer to a 16-bit version).

In general, Equation 2.27 is difficult to compute since it requires evaluation of complex exponents. The result of a complex exponent is a complex number, thus the implementation requires representing functions, operations and variables for storage and processing using complex numbers. Although operations with complex numbers are well supported in mathematical packages, this equation obscures the true interpretation of the Fourier definition. Fortunately, by algebraic manipulation we can rewrite Equation 2.27 in a form that separates the imaginary and real parts. By recalling Euler's formula  $e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$ , then Equation 2.27 becomes

$$FP_{u,v} = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} P_{x,y} \left( \cos\left(\frac{2\pi}{M}vy\right) - j\sin\left(\frac{2\pi}{M}vy\right) \right) \left( \cos\left(\frac{2\pi}{N}ux\right) - j\sin\left(\frac{2\pi}{N}ux\right) \right) \quad (2.28)$$

By grouping terms, we have that

$$FP_{u,v} = \frac{1}{MN} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} P_{x,y} \left( \left( \cos\left(\frac{2\pi}{N}ux\right) \cos\left(\frac{2\pi}{M}vy\right) - \sin\left(\frac{2\pi}{N}ux\right) \sin\left(\frac{2\pi}{M}vy\right) \right) - j \left( \cos\left(\frac{2\pi}{N}ux\right) \sin\left(\frac{2\pi}{M}vy\right) + \sin\left(\frac{2\pi}{N}ux\right) \cos\left(\frac{2\pi}{M}vy\right) \right) \right) \quad (2.29)$$

Equations 2.27 and 2.28 represent the same transform, but Equation 2.28 permits computation of the real and imaginary parts as trigonometric functions. More importantly, the trigonometric form shows how the values of  $u$  and  $v$  define the frequency used in the transform. This equation also shows the symmetry of the transform; since  $\sin(-t) = -\sin(t)$  and  $\cos(-t) = \cos(t)$ , then the magnitude of the transform defined in Equation 2.7 is symmetrical at the origin.

```

for u in range(-maxFreqW, maxFreqW + 1):
    entryW = u + maxFreqW

    for v in range(-maxFreqH, maxFreqH + 1):
        entryH = v + maxFreqH
        coeff[entryH, entryW] = [0, 0]

        for x in range(0, width):
            sumY = [0, 0]

            for y in range(0, height):
                sumY[0] += inputImage[y,x] * cos(y * wh * v)
                sumY[1] += inputImage[y,x] * sin(y * wh * v)
            coeff[entryH, entryW][0] += sumY[0]*cos(x*ww*u) - sumY[1]*sin(x*ww*u)
            coeff[entryH, entryW][1] -= cos(x*ww*u)*sumY[1] + sin(x * ww*u) * sumY[0]
    
```

### Code 2.2 2D DFT, Trigonometric Form

The implementation of the trigonometric form is shown in Code 2.2. To illustrate separability, the implementation computes the inner and outer summations in Equation 2.28, though the same result would be obtained if the real and imaginary parts were computed by using Equation 2.29. In any case, the implementation should perform the sum for the real and imaginary parts by evaluating sines and cosines at different frequencies. In the code, the array `sumY[0]` and `coeff[u,v][0]` store the real values and `sumY[1]` and `coeff[u,v][1]` the imaginary values. Finally the coefficients need to be scaled by  $\frac{1}{MN}$  according to Equation 2.28.

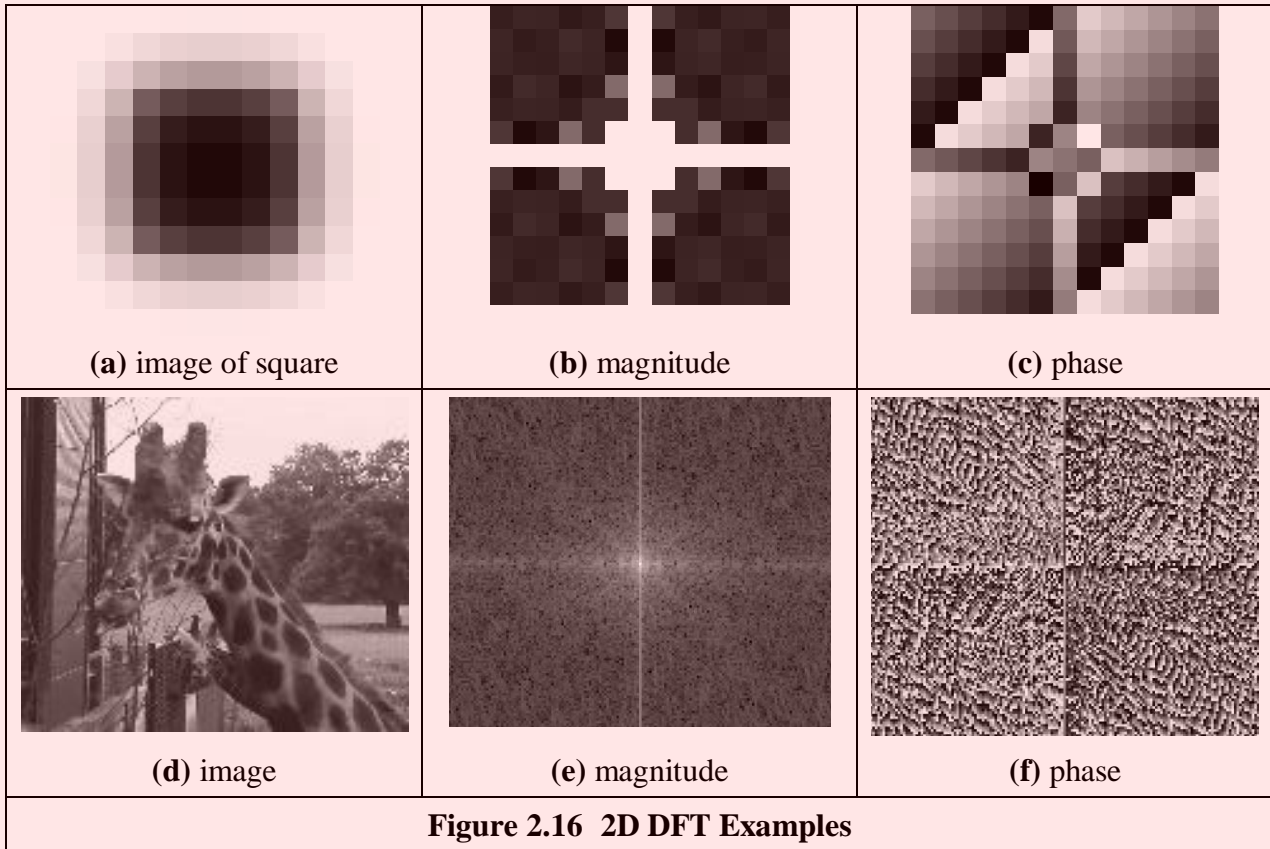
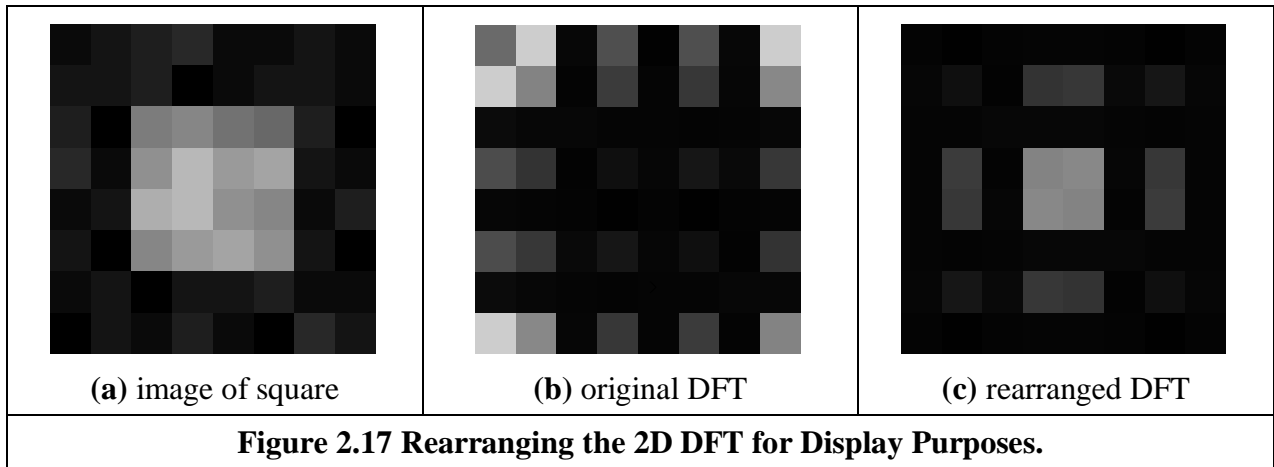


Figure 2.16 shows two examples of the DFT obtained with Code 2.2. The results of the Fourier expansion are not presented by showing the imaginary and real values of  $\mathbf{FP}_{u,v}$ , but the complex values are shown as magnitude and phase (Equations 2.7 and 2.8). Note the symmetries in the magnitude and phase that result from the symmetries of the cosine and sine functions.

Equation 2.29 reveals the nature of the Fourier transform as a description of an image using frequency; each value  $\mathbf{FP}_{u,v}$  defines sine and cosine waves at a given frequency. The values of  $u$  and  $v$  produce waves with frequency that increases as these values increase. As such, the position of each component reflects its frequency: low frequency components are near the origin and high frequency components are further away. Notice also that frequencies can be positive and negative and the lowest frequency component, for zero frequency – the d.c. component – represents the average value of the samples. In Code 2.2, we used this interpretation where  $u$  and  $v$  represent frequencies. As such, the loops in the implementation iterate over a frequency range. These ranges can be arbitrarily selected to obtain only the fine detail by just computing high frequencies or to obtain coarse features by choosing low frequencies. As we discussed in Section 2.4, the maximum frequency we can use is given by the half of image size (or the result will contain aliasing). In the implementation, the result of the summation for each pair  $u, v$  is stored in a cell in the output array `coeff`. The index to the cell is stored in the variables `entryW` and `entryH`. These values are set such that the zero frequency is stored in the centre of the output array and the negative frequencies are to the left of and down from the centre position. This way to select the output cells is just a standard way to organise and visualise the frequencies. It is important to notice that the loops can be changed to iterate over the output array (since Equations 2.29 and 2.27 are the same transform), but iterating in frequency gives a clear meaning to the frequency content in the output.

Code 2.1 shows the origin of the transform (low frequency components) at the corners of the transform. The image of the square in 2.17(a) produces the result for 2.17(b). A spatial transform is easier to visualize if the d.c. (zero frequency) component is in the centre, with frequency increasing towards the edge of the image. This can be arranged either by rotating each of the four

quadrants in the Fourier transform by 180°. An alternative is to *reorder* the original image to give a transform which has been shifted to the centre. Both operations result in the image in 2.17(c) wherein the transform is much more easily seen. Note that this is aimed to improve visualization and does not change any of the frequency domain information, only the standard way it is displayed.



To rearrange the image so that the d.c. component is in the centre, the frequency components need to be reordered. This can be achieved simply by multiplying each image point  $\mathbf{P}_{x,y}$  by  $-1^{(x+y)}$ . Since  $\cos(-\pi) = -1$ , then  $-1 = e^{-j\pi}$  (the minus sign is introduced just to keep the analysis neat) so we obtain the transform of the multiplied image as:

$$\begin{aligned}
 \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)} \times -1^{(x+y)} &= \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)(ux+vy)} \times e^{-j\pi(x+y)} \\
 &= \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} e^{-j\left(\frac{2\pi}{N}\right)\left(\left(u+\frac{N}{2}\right)x+\left(v+\frac{N}{2}\right)y\right)} \\
 &= \mathbf{FP}_{u+\frac{N}{2},v+\frac{N}{2}}
 \end{aligned} \tag{2.30}$$

According to Equation 2.30, when pixel values are multiplied by  $-1^{(x+y)}$  the Fourier transform becomes shifted along each axis by half the number of samples. According to the replication theorem, Equation 2.26, the transform replicates along the frequency axes. This implies that the centre of a transform image will now be the d.c. component. (Another way of interpreting this is that rather than look at the frequencies centred on where the image is, our viewpoint has been shifted so as to be centred on one of its corners - thus invoking the replication property.) This brings equivalence between the trigonometric form (for the magnitude see Figures 2.16(b) and (d)) and the exponential form (see Figure 2.17(b)). The operator `Rearrange`, in Code 2.3, is used prior to transform calculation and leads to the image of Figure 2.17(c), and all later transform images.

```

function rearranged = rearrange(image)
%get dimensions
[rows,cols]=size(image);

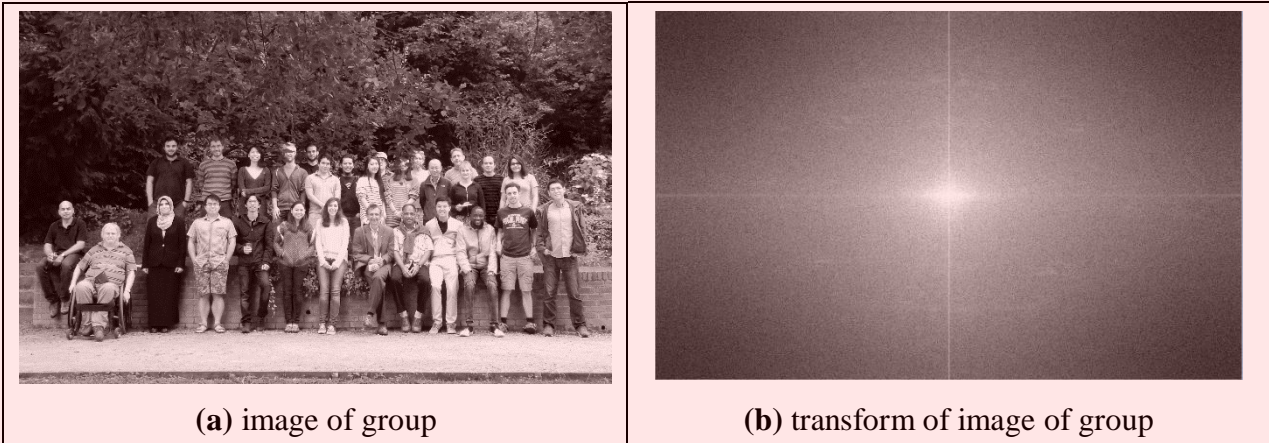
%rearrange image
for x = 1:cols %address all columns
  for y = 1:rows %address all rows
    rearranged(y,x)=image(y,x)*((-1)^(y+x)); %Eq. 2.30
  end
end

```

end

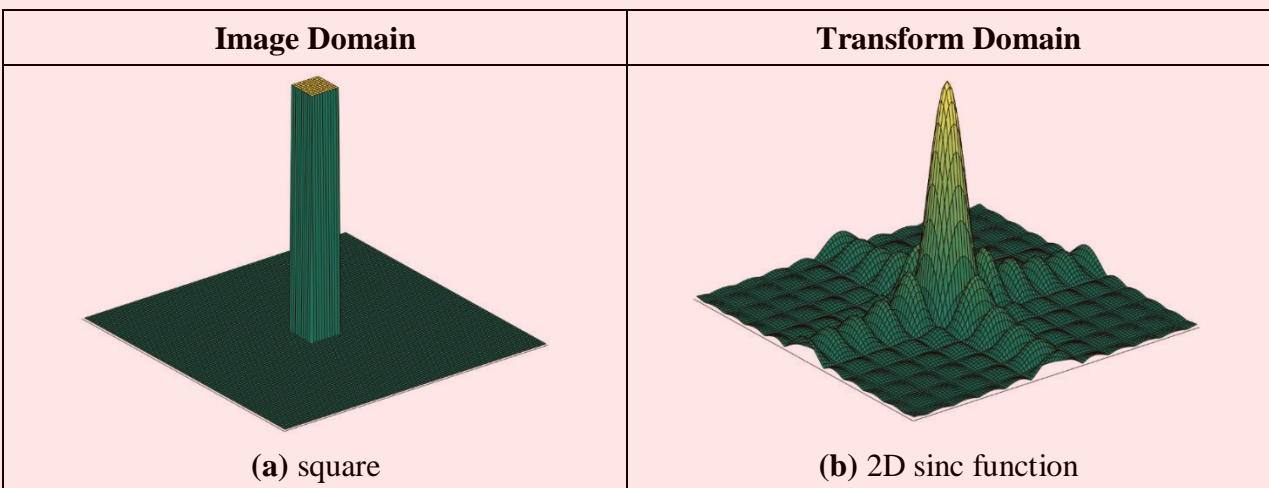
**Code 2.3 Reordering for Transform Calculation**

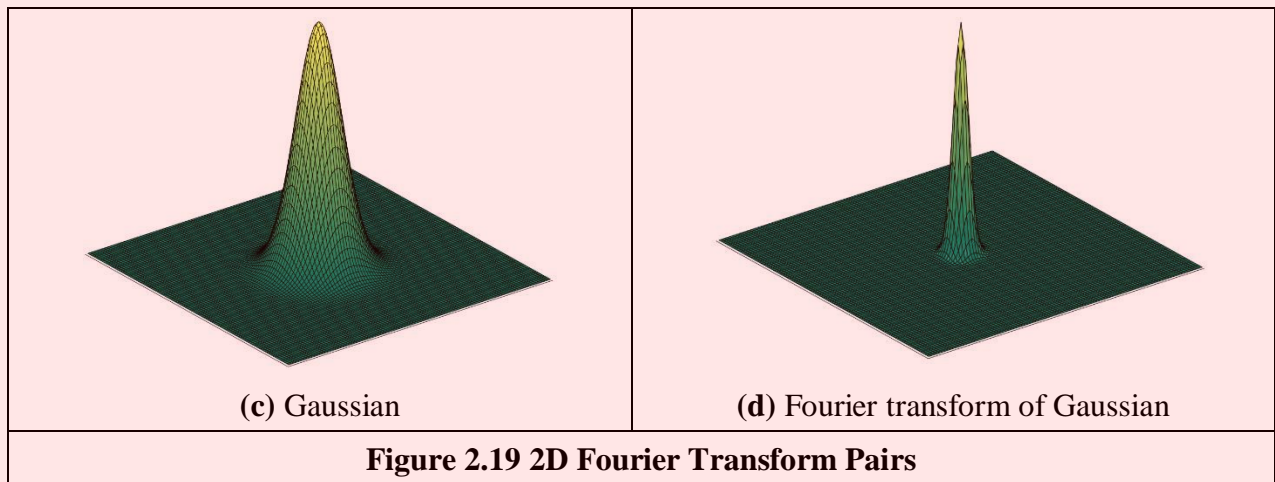
The full effect of the Fourier transform is shown by application to an image of much higher resolution. Figure 2.18(a) shows the image of a group of people and Figure 2.18(b) shows its transform. The transform reveals that much of the information is carried in the lower frequencies since this is where most of the spectral components concentrate. This is because the image has many regions where the brightness does not change a lot, such as in the foliage. The high frequency components reflect change in intensity. Accordingly, the higher frequency components arise from things that change fast and from the borders of objects.



**Figure 2.18 Applying the Fourier Transform to the Image of a Group of People**

As with the 1D Fourier transform, there are 2D Fourier transform pairs, illustrated in Figure 2.19. The 2D Fourier transform of a two dimensional pulse, Figure 2.19(a), is a two dimensional sinc function, in Figure 2.19(b). The 2D Fourier transform of a Gaussian function, in Figure 2.19(c), is again a two dimensional Gaussian function in the frequency domain, in Figure 2.19(d).





## 2.6 Properties of the Fourier Transform

### 2.6.1 Shift Invariance

The decomposition into spatial frequency does not depend on the position of features within the image. If we shift all the features by a fixed amount, or acquire the image from a different position, the magnitude of its Fourier transform does not change. This property is known as *shift invariance*. By denoting the delayed version of  $p(t)$  as  $p(t - \tau)$ , where  $\tau$  is the delay, and the Fourier transform of the shifted version as  $\mathfrak{F}[p(t - \tau)]$ , we obtain the relationship between a time domain shift in the time and frequency domains as:

$$\mathfrak{F}[p(t - \tau)] = e^{-j\omega\tau} P(\omega) \quad (2.31)$$

Accordingly, the magnitude of the Fourier transform is:

$$|\mathfrak{F}[p(t - \tau)]| = |e^{-j\omega\tau} P(\omega)| = |e^{-j\omega\tau}| |P(\omega)| = |P(\omega)| \quad (2.32)$$

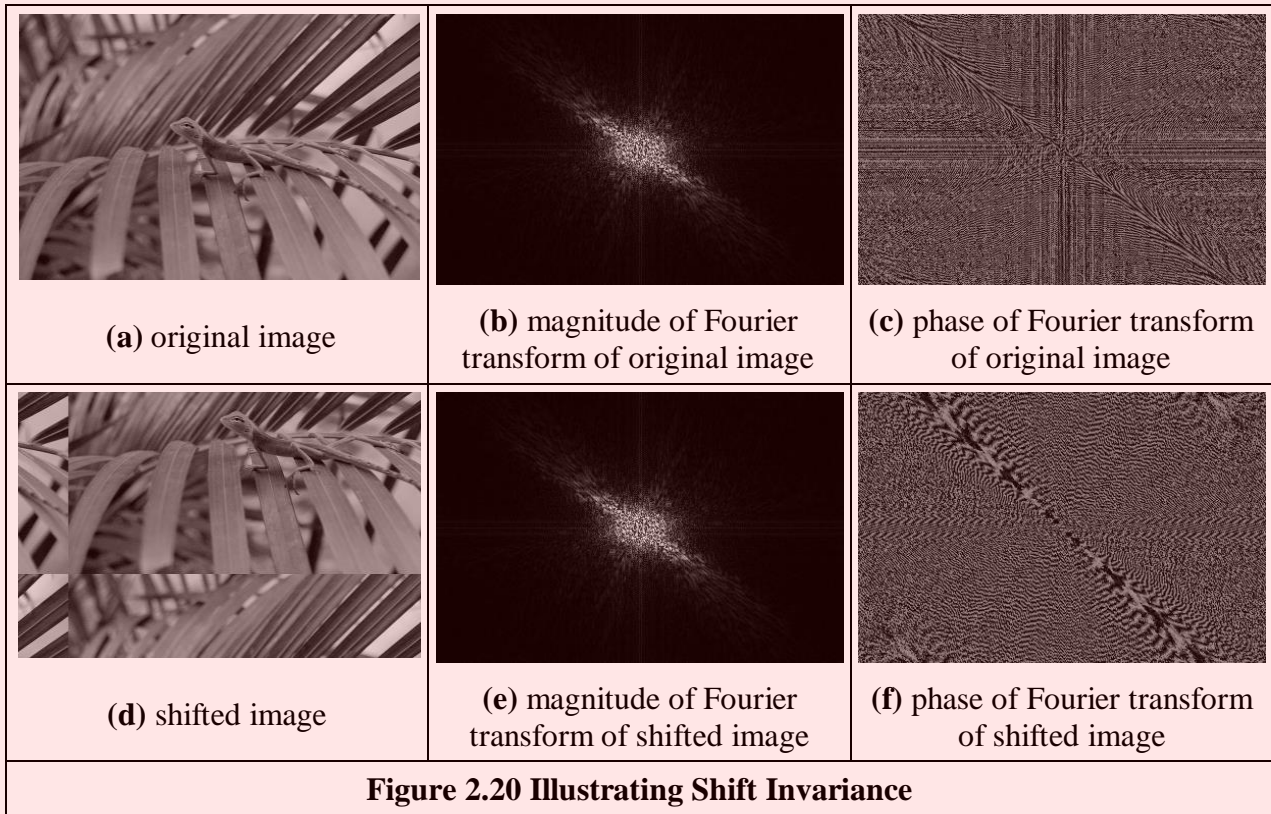
and since the magnitude of the exponential function is 1.0 then the magnitude of the Fourier transform of the shifted image equals that of the original (unshifted) version. We shall use this property later in Chapter 7 when we use Fourier theory to describe shapes. There, it will allow us to give the same description to different instances of the same shape, but a different description to a different shape. You do not get something for nothing: even though the magnitude of the Fourier transform remains constant, its phase does not. The phase of the shifted transform is:

$$\arg(\mathfrak{F}[p(t - \tau)]) = \arg(e^{-j\omega\tau} P(\omega)) \quad (2.33)$$

The implementation of a `shift` operator, Code 2.4, uses the modulus operation `%` to enforce the cyclic shift. The `shiftDistance` is a parameter that defines the length of the horizontal shift along the  $x$  axis.

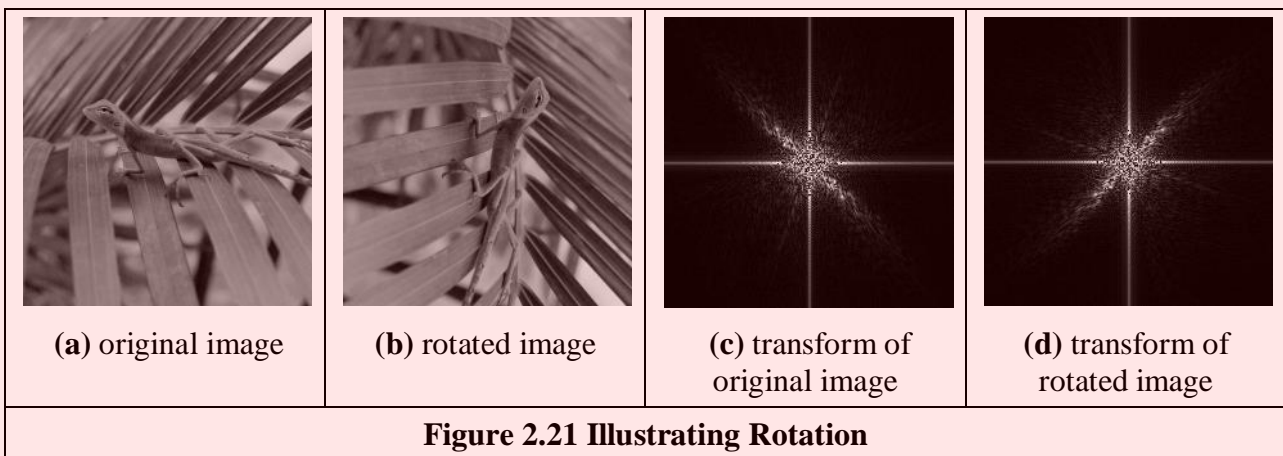
```
for x,y in itertools.product(range(0, width), range(0, height)):
    xShift = (x - shiftDistance) % width
    shiftImage[y,x] = inputImage[y,xShift]
```

**Code 2.4 Shifting an Image**



This process is illustrated in Figure 2.20. An original image, Figure 2.20(a), is shifted along the  $x$  and the  $y$  axes, Figure 2.20(d). The shift is cyclical, so parts of the image wrap around; those parts at the top of the original image appear at the base of the shifted image. The Fourier transform of the original image and of the shifted image are identical: Figure 2.20(b) appears the same as Figure 2.20(e). The phase differs: the phase of the original image Figure 2.20(c) is clearly different from the phase of the shifted image, Figure 2.20(f).

The differing phase implies that, in application, the magnitude of the Fourier transform of a face, say, will be the same irrespective of the position of the face in the image (i.e. the camera or the subject can move up and down), assuming that the face is much larger than its image version. This implies that if the magnitude of the Fourier transform is used to analyse an image of a human face or one of cloth, to describe it by its spatial frequency, we do not need to control the position of the camera, or the object, precisely.



### 2.6.2 Rotation

The Fourier transform of an image *rotates* when the source image rotates. This is to be expected since the decomposition into spatial frequency reflects the orientation of features within the image. As such, orientation dependency is built into the Fourier transform process.

This implies that if the frequency domain properties are to be used in image analysis, via the Fourier transform, the orientation of the original image needs to be known, or fixed. It is often possible to fix orientation, or to estimate its value when a feature's orientation cannot be fixed. Alternatively, there are techniques to impose invariance to rotation, say by translation to a polar representation, though this can prove to be complex.

The effect of rotation is illustrated in Figure 2.21. An image, Figure 2.21(a), is rotated by 90° to give the image in Figure 2.21(b). Comparison of the transform of the original image, Figure 2.21(c), with the transform of the rotated image, Figure 2.21(d) shows that the transform has been rotated by 90°, by the same amount as the image. In fact, close inspection of Figures 2.21(c) and (d) shows that the diagonal axis is consistent with the normal to the axis of the leaves (where the change mainly occurs), and this is the axis that rotates.

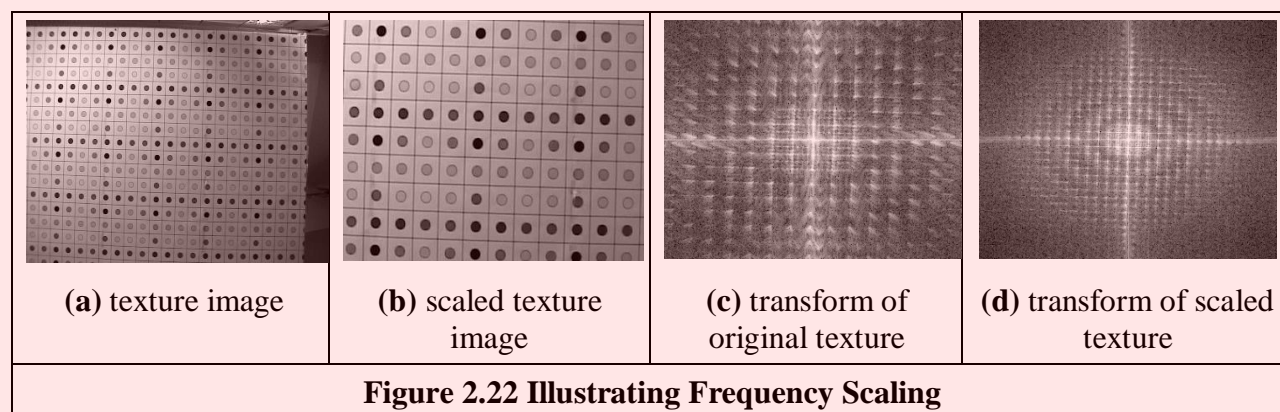
### 2.6.3 Frequency Scaling

By definition, time is the reciprocal of frequency. So if an image is compressed, equivalent to reducing time, its frequency components will spread, corresponding to increasing frequency. Mathematically the relationship is that the Fourier transform of a function of time multiplied by a scalar  $\lambda$ ,  $p(\lambda t)$ , gives a frequency domain function  $P(\omega/\lambda)$ , so:

$$\mathfrak{F}[p(\lambda t)] = \frac{1}{\lambda} P\left(\frac{\omega}{\lambda}\right) \tag{2.34}$$

This is illustrated in Figure 2.22 where the image of spots (a 3D calibration target), Figure 2.22(a), is reduced in scale, Figure 2.22(b), thereby increasing the spatial frequency. The DFT of the original image is shown in Figure 2.22(c) which reveals that the large spatial frequencies in the original image are arranged in a star-like pattern. As a consequence of scaling the original image, the spectrum will spread from the origin consistent with an increase in spatial frequency, as shown in Figure 2.22(d). This retains the star-like pattern, but with points at a greater distance from the origin.

The implications of this property are that if we reduce the scale of an image, say by imaging at a greater distance, we will alter the frequency components. The relationship is linear: the amount of reduction, say the proximity of the camera to the target, is directly proportional to the scaling in the frequency domain.



### 2.6.4 Superposition (Linearity)

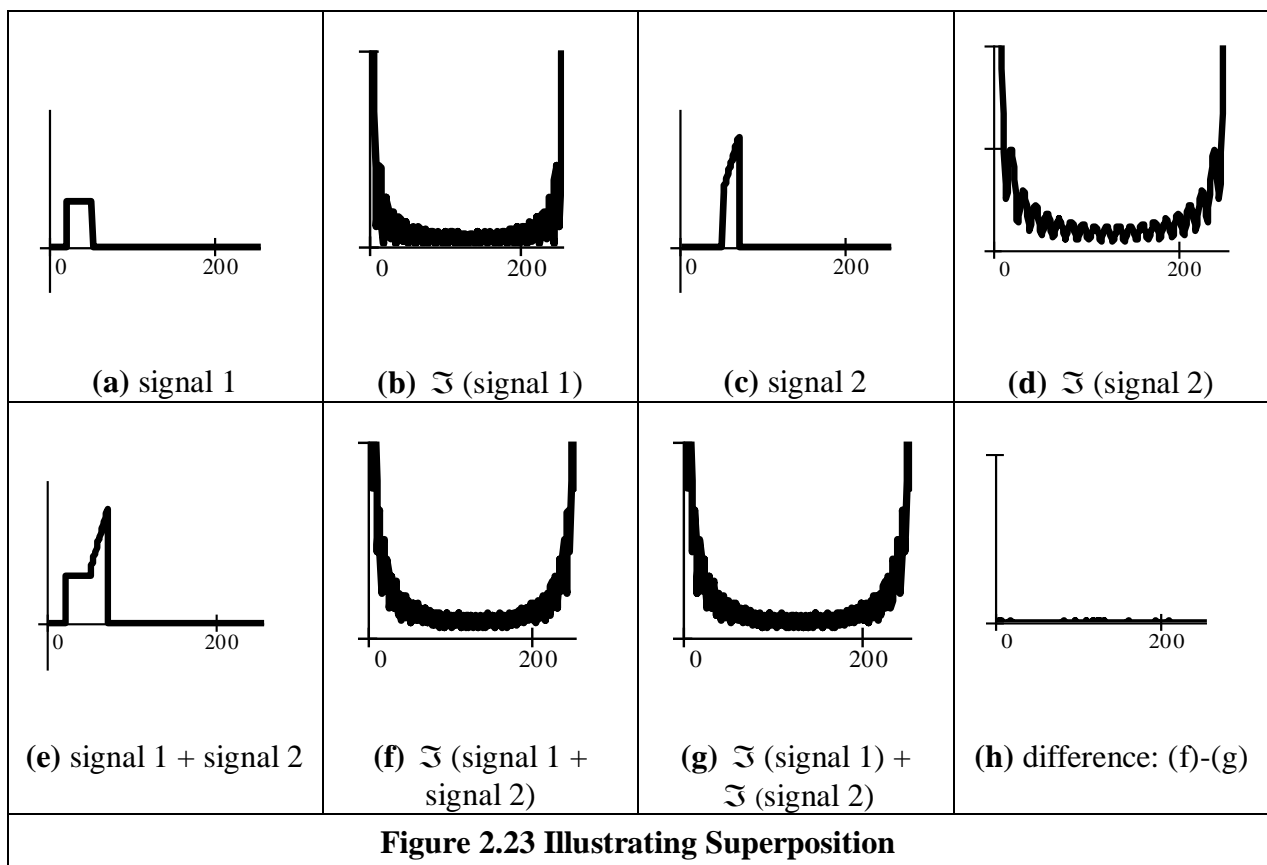
The *principle of superposition* is very important in systems analysis. Essentially, it states that a system is linear if its response to two combined signals equals the sum of the responses to the individual signals. Given an output  $O$  which is a function of two inputs  $I_1$  and  $I_2$ , the response to signal  $I_1$  is  $O(I_1)$ , that to signal  $I_2$  is  $O(I_2)$ , and the response to  $I_1$  and  $I_2$ , when applied together, is  $O(I_1 + I_2)$ , the superposition principle states:

$$O(I_1 + I_2) = O(I_1) + O(I_2) \tag{2.35}$$

Any system which satisfies the principle of superposition is termed *linear*. The Fourier transform is a linear operation since, for two signals  $p_1$  and  $p_2$ :

$$\mathfrak{F}[p_1 + p_2] = \mathfrak{F}[p_1] + \mathfrak{F}[p_2] \tag{2.36}$$

In application this suggests that we can separate images by looking at their frequency domain components. This is illustrated for one-dimensional signals in Figure 2.23. One signal is shown in Figure 2.23(a) and a second is shown in Figure 2.23(c). The Fourier transforms of these signals are shown in Figures 2.23(b) and (d). The addition of these signals is shown in Figure 2.23(e) and its transform in Figure 2.23(f). The Fourier transform of the added signals differs little from the addition of their transforms, Figure 2.23(g). This is confirmed by subtraction of the two, Figure 2.23(h) (some slight differences can be seen, but these are due to numerical error).



By way of example, given the image of a fingerprint in blood on cloth it is very difficult to separate the fingerprint from the cloth by analysing the combined image. However, by translation to the frequency domain, the Fourier transform of the combined image shows strong components due to



the texture (this is the spatial frequency of the cloth’s pattern) and weaker, more scattered, components due to the fingerprint. If we suppress the frequency components due to the cloth’s texture, and invoke the inverse Fourier transform, then the cloth will be removed from the original image. The fingerprint can now be seen in the resulting image.

### 2.6.5 The Importance of Phase

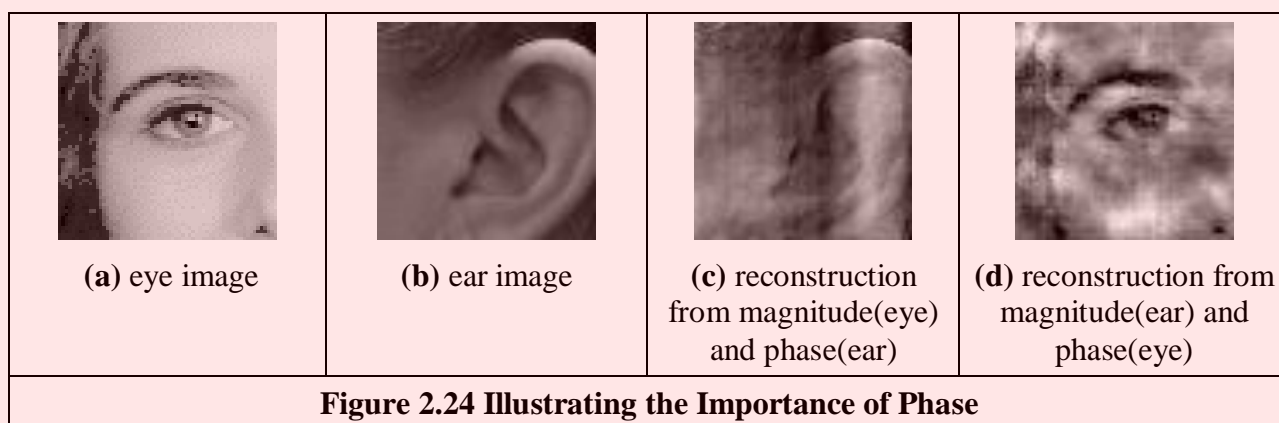
You might reasonably ask: “what is the importance of phase?”. It is actually a pretty reasonable question. *Phase* is about how signals are arranged (or add up), whereas magnitude is about how big the signals are. As such, one might intuitively think that the magnitude is more important than the phase. It is more complex than this: phase can actually be viewed to be more important (though you need both magnitude and phase to reconstruct a signal). To illustrate this we shall take two images, one of the eye and the other of an ear (OK, ears are rather ugly but they are unique to their owner) as shown in Figure 2.24. Here we have reconstructed images by taking the magnitude of one image and the phase (the argument) of another. From Eq 2.6 we have

$$\mathbf{Fp}(\omega) = \text{Re}(\mathbf{Fp}(\omega)) + j \text{Im}(\mathbf{Fp}(\omega)) = \text{magnitude} \times \cos(\text{phase}) + j \times \text{magnitude} \times \sin(\text{phase}) \quad (2.37)$$

where magnitude and phase are calculated according to Equations 2.7 and 2.8, respectively. The rearranged Fourier transform is then

$$\mathbf{Fp}(\omega) = |\mathbf{Fp}(\text{image 1})| \times \cos(\text{arg}(\mathbf{Fp}(\text{image 2}))) + j \times |\mathbf{Fp}(\text{image 1})| \times \sin(\text{arg}(\mathbf{Fp}(\text{image 2})))$$

When image 1 is the eye and image 2 is the ear then the reconstructed image (via the inverse FT) looks most like the image of the ear, Figure 2.24(c); when it is the other way round, the image is much closer to the eye, Figure 2.24(d). So it is the phase that is controlling the reconstruction in this case, not the magnitude (the bit represented by the magnitude hardly shows in the reconstructions here). Clearly the phase is very important in the representation of a signal.



## 2.7 Transforms other than Fourier

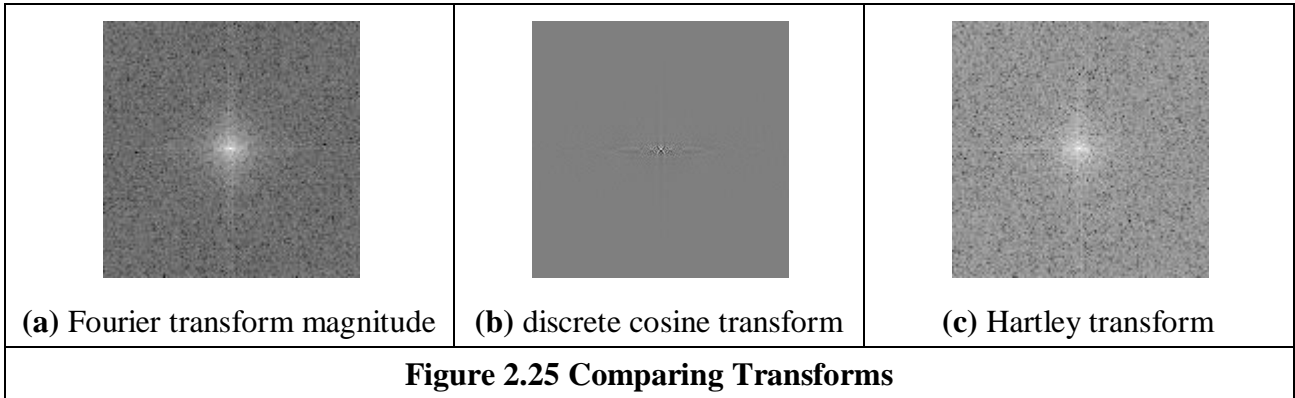
### 2.7.1 Discrete Cosine Transform

The *Discrete Cosine Transform* (DCT) [Ahmed74] is a real transform that has great advantages in energy compaction. Its definition for spectral components  $\mathbf{DP}_{u,v}$  is:

$$\mathbf{DP}_{u,v} = \begin{cases} \frac{1}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} & \text{if } u = 0 \text{ and } v=0 \\ \frac{2}{N^2} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} \times \cos\left(\frac{(2x+1)u\pi}{2N}\right) \times \cos\left(\frac{(2y+1)v\pi}{2N}\right) & \text{otherwise} \end{cases} \quad (2.38)$$

There are many variants of the definition of the DCT and we are concerned only with principles here. The inverse DCT is defined by

$$\mathbf{P}_{x,y} = \frac{1}{N^2} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \mathbf{DP}_{u,v} \times \cos\left(\frac{(2x+1)u\pi}{2N}\right) \times \cos\left(\frac{(2y+1)v\pi}{2N}\right) \quad (2.39)$$



A fast version of the DCT is available, like the FFT, and calculation can be based on the FFT. Both implementations offer about the same speed. The Fourier transform is not actually optimal for *image coding* since the Discrete Cosine transform can give a higher compression rate, for the same image quality. This is because the cosine basis functions can afford for high energy compaction. This can be seen by comparison of Figure 2.25(b) with Figure 2.25(a), which reveals that the DCT components are much more concentrated around the origin, than those for the Fourier Transform. This is the compaction property associated with the DCT. The DCT has actually been considered as optimal for image coding, and this is why it is found in the JPEG and MPEG standards for coded image transmission.

The DCT is actually shift variant, due to its cosine basis functions. In other respects, its properties are very similar to the DFT, with one important exception: it has not yet proved possible to implement convolution with the DCT. It is actually possible to calculate the DCT via the FFT. This has been performed in Figure 2.25(b).

The Fourier transform essentially decomposes, or decimates, a signal into sine and cosine components, so the natural partner to the DCT is the *Discrete Sine Transform* (DST). However, the DST transform has odd basis functions (sine) rather than the even ones in the DCT. This lends the DST transform some less desirable properties, and it finds much less application than the DCT.

### 2.7.2 Discrete Hartley Transform

The *Hartley transform* [Hartley42] is a form of the Fourier transform, but without complex arithmetic, with result for the face image shown in Figure 2.25(c). Oddly, though it sounds like a very rational development, the Hartley transform was first invented in 1942, but not rediscovered and then formulated in discrete form until 1983 [Bracewell83]. One advantage of the Hartley transform is that the forward and inverse transform is the same operation; a disadvantage is that phase is built into the order of frequency components since it is not readily available as the argument

of a complex number. The definition of the Discrete Hartley Transform (DHT) replaces the exponent basis by the cas function defined as

$$\text{cas}(t) = \cos(t) + \sin(t) \quad (2.40)$$

Thus the transform components  $\mathbf{HP}_{u,v}$  are:

$$\mathbf{HP}_{u,v} = \frac{1}{\sqrt{MN}} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} \mathbf{P}_{x,y} \left( \cos\left(\frac{2\pi}{M}vy\right) + \sin\left(\frac{2\pi}{M}vy\right) \right) \left( \cos\left(\frac{2\pi}{N}ux\right) + \sin\left(\frac{2\pi}{N}ux\right) \right) \quad (2.41)$$

For a square image, this can be written as

$$\mathbf{HP}_{u,v} = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{P}_{x,y} \left( \sin\left(\frac{2\pi}{N}(ux + vy)\right) + \cos\left(\frac{2\pi}{N}(ux - vy)\right) \right) \quad (2.42)$$

The inverse Hartley transform is the same process, but applied to the transformed image.

$$\mathbf{P}_{x,y} = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \mathbf{HP}_{x,y} \left( \sin\left(\frac{2\pi}{N}(ux + vy)\right) + \cos\left(\frac{2\pi}{N}(ux - vy)\right) \right) \quad (2.43)$$

The implementation is then the same for both the forward and the inverse transforms. Again, a fast implementation is available - the *Fast Hartley Transform* [Bracewell84] (though some suggest that it should be called the Bracewell transform, eponymously). It is actually possible to calculate the DFT of a function,  $F(u)$ , from its Hartley transform,  $H(u)$ . The analysis here is based on 1-dimensional data, but only for simplicity since the argument extends readily to two dimensions. By splitting the Hartley transform into its odd and even parts,  $O(u)$  and  $E(u)$ , respectively we obtain:

$$H(u) = O(u) + E(u) \quad (2.44)$$

where:

$$E(u) = \frac{H(u) + H(N-u)}{2} \quad (2.45)$$

and

$$O(u) = \frac{H(u) - H(N-u)}{2} \quad (2.46)$$

The DFT can then be calculated from the DHT simply by

$$F(u) = E(u) - j \times O(u) \quad (2.47)$$

Conversely, the Hartley transform can be calculated from the Fourier transform by:

$$H(u) = \text{Re}[F(u)] - \text{Im}[F(u)] \quad (2.48)$$

where  $\text{Re}[\ ]$  and  $\text{Im}[\ ]$  denote the real and the imaginary parts, respectively. This emphasises the natural relationship between the Fourier and the Hartley transform. The image of Figure 2.25(c) has been calculated via the 2D FFT using Equation 2.48. Note that the transform in Figure 2.25(c) is the complete transform whereas the Fourier transform in Figure 2.25(a) shows magnitude only. Naturally, as with the DCT, the properties of the Hartley transform mirror those of the Fourier transform. Unfortunately, the Hartley transform does not have shift invariance but there are ways to handle this. Also, convolution requires manipulation of the odd and even parts.

Code 2.5 illustrates the computation of the Hartley transform in Equation 2.28. Similar to Code 2.2 the values are defined for a range of frequencies and the result is processed such that the zero frequency is at the centre of the output array. Notice that this equation is also separable, so it is possible to follow a similar implementation of the Fourier transform by maintaining the first cosine

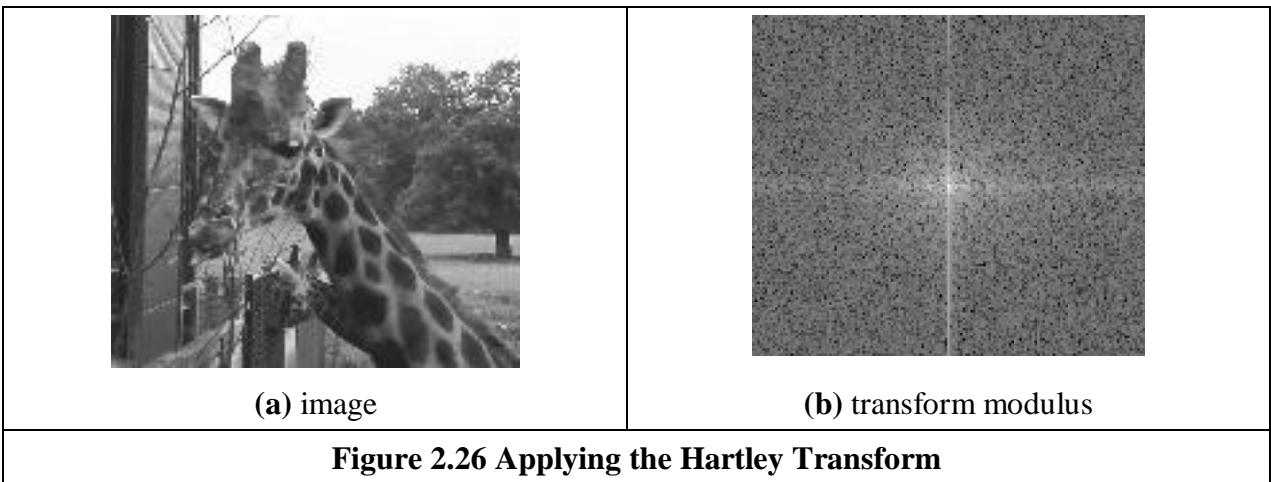
and sine sum in one direction and then performing the multiplication with the terms in the other direction. In this implementation, the components are computed by combining the product of both directions.

```

for u in range(-maxFreqW, maxFreqW + 1):
    entryW = u + maxFreqW
    for v in range(-maxFreqH, maxFreqH + 1):
        entryH = v + maxFreqH
        for x,y in itertools.product(range(0, width), range(0, height)):
            coeff[entryH, entryW] += inputImage[y,x] * \
                (cos(x*ww*u) + sin(x*ww*u)) * (cos(y*wh*v) + sin(y*wh*v))
    
```

**Code 2.5 Hartley Transform**

Figure 2.26 shows an example of the transform obtained using Code 2.5. Figure 2.26 (a) shows the input image and Figure 2.26 (b) shows the result of the transform. The resulting components have positive and negative values, so to show the results in an image it is necessary to perform some normalisation. The image shown in Figure 2.26 (b) shows the log of the absolute value, so it is comparable to the Fourier magnitude. Notice that the transform is symmetrical and it is rather similar to the results in Figure 2.16(e). This is because both, the Fourier and Hartley transforms decompose the image into frequency components.



### 2.7.3 Introductory Wavelets

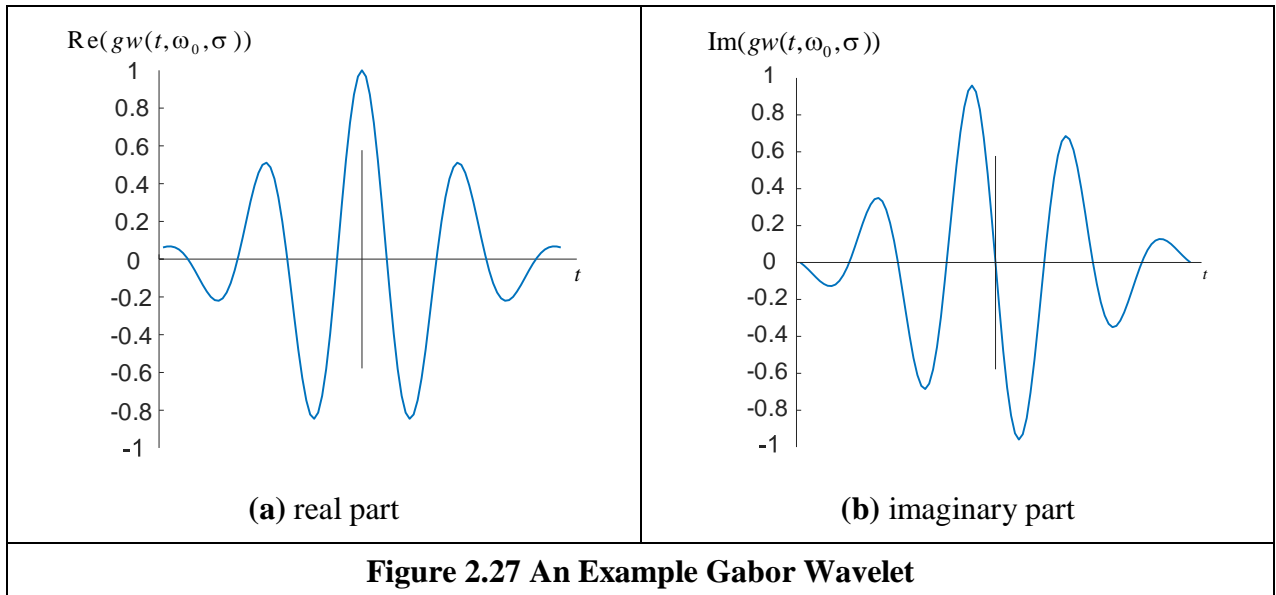
#### 2.7.3.1 Gabor Wavelet

*Wavelets* are more recent approach to signal processing than the Fourier transform, being introduced only in the nineties [Daubechies90]. Their main advantage is that they allow multi-resolution analysis (analysis at different scales, or resolution). Furthermore, wavelets allow decimation in space and frequency, simultaneously. Earlier transforms actually allow decimation in frequency, in the forward transform, and in time (or position) in the inverse. In this way, the Fourier transform gives a measure of the frequency content of the whole image: the contribution of the image to a particular frequency component. Simultaneous decimation allows us to describe an image in terms of frequency which occurs at a position, as opposed to an ability to measure frequency content across the whole image. Clearly this gives us a greater descriptive power, which can be used to good effect.

First though, we need a basis function, so that we can decompose a signal. The basis functions in the Fourier transform are sinusoidal waveforms at different frequencies. The function of the Fourier transform is to convolve these sinusoids with a signal to determine how much of each is present. The *Gabor wavelet* is well suited to introductory purposes, since it is essentially a sinewave modulated by a Gaussian envelope. The Gabor wavelet  $gw$  is given by

$$gw(t, \omega_0, \sigma) = e^{-j\omega_0 t} e^{-\left(\frac{t-t_0}{\sigma}\right)^2} \tag{2.49}$$

where  $\omega_0 = 2\pi f_0$  is the modulating frequency,  $t_0$  dictates position and  $\sigma$  controls the width of the Gaussian envelope which embraces the oscillating signal. An example Gabor wavelet is shown in Figure 2.27 which shows the real and the imaginary parts (the modulus is the Gaussian envelope). Increasing the value of  $\omega_0$  increases the frequency content within the envelope whereas increasing the value of  $\sigma$  spreads the envelope without affecting the frequency. So why does this allow simultaneous analysis of time and frequency? Given that this function is the one convolved with the test data, then we can compare it with the Fourier transform. In fact, if we remove the term on the right hand side of Equation 2.49, we return to the sinusoidal basis function of the Fourier transform, the exponential in Equation 2.1. Accordingly, we can return to the Fourier transform by setting  $\sigma$  to be very large. Alternatively, setting  $f_0$  to zero removes frequency information. Since we operate in between these extremes, we obtain position and frequency information simultaneously.



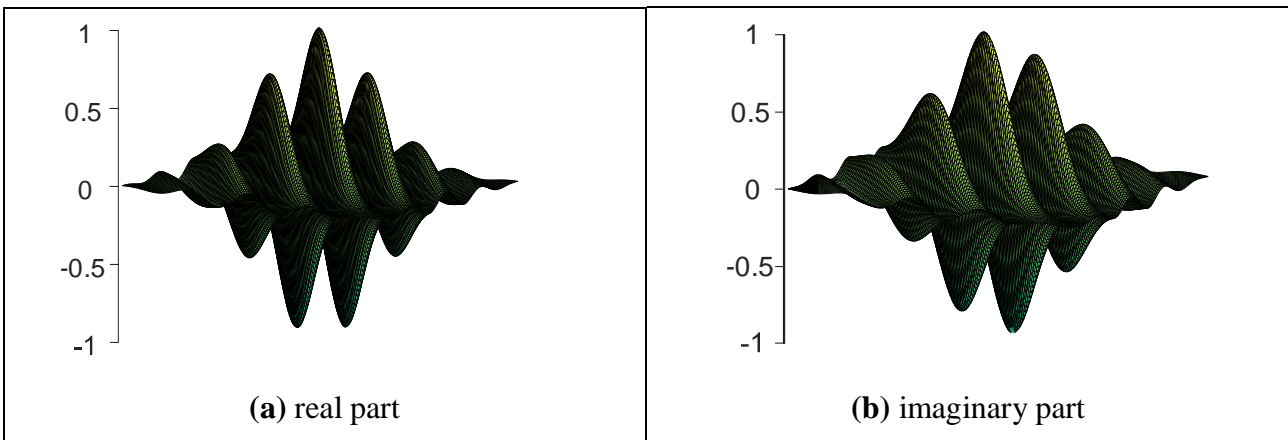
**Figure 2.27 An Example Gabor Wavelet**

Actually, an infinite class of wavelets exists which can be used as an expansion basis in signal decimation. One approach [Daugman88] has generalised the Gabor function to a 2D form aimed to be optimal in terms of spatial and spectral resolution. These 2D Gabor wavelets are given by

$$gw2D(x, y, \omega_0, \sigma) = \frac{1}{\sigma\sqrt{\pi}} e^{-\left(\frac{(x-x_0)^2+(y-y_0)^2}{2\sigma^2}\right)} e^{-j\omega_0((x-x_0)\cos(\theta)+(y-y_0)\sin(\theta))} \tag{2.50}$$

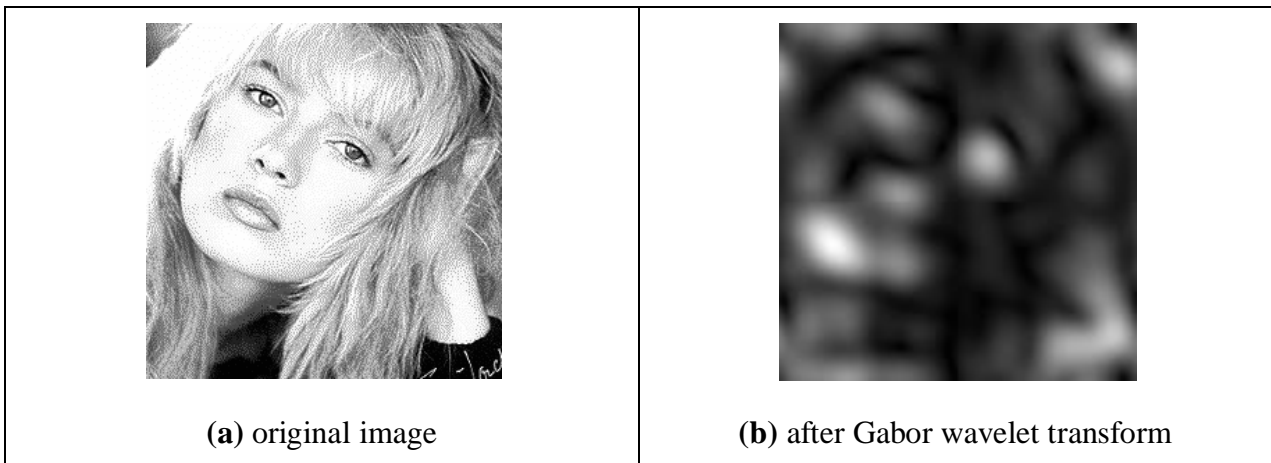
where  $x_0$  and  $y_0$  control position,  $\omega_0 = 2\pi f_0$  controls the frequency of modulation along either axis, and  $\theta$  controls the direction (orientation) of the wavelet (as implicit in a two dimensional system). Naturally, the shape of the area imposed by the 2D Gaussian function could be elliptical if different variances were allowed along the  $x$  and  $y$  axes (the frequency can also be modulated

differently along each axis). Figure 2.28, of an example 2D Gabor wavelet, shows that the real and imaginary parts are even and odd functions, respectively; again, different values for  $f_0$  and  $\sigma$  control the frequency and envelope's spread respectively, the extra parameter  $\theta$  controls rotation.



**Figure 2.28 Example 2-Dimensional Gabor Wavelet**

The function of the wavelet transform is to determine where and how each wavelet specified by the range of values for each of the free parameters occurs in the image. Clearly, there is a wide choice that depends on application. An example transform is given in Figure 2.29. Here, the Gabor wavelet parameters have been chosen in such a way as to select face features: the eyes, nose and mouth have come out very well. (Not that it's anything to do with Gabor, but the lady in the image had an interesting career: she started as lady of the night, went on to be a pop star and then wrote a book about it. In her career she had a lot of positions, especially at the start.) These features are where there is local frequency content with orientation according to the head's inclination. Naturally, these are not the only features with these properties, the cuff of the sleeve is highlighted too! But this does show the Gabor wavelet's ability to select and analyse localised variation in image intensity.



**Figure 2.29 An Example Gabor Wavelet Transform**

However, the conditions under which a set of continuous Gabor wavelets will provide a complete representation of any image (i.e. that any image can be reconstructed) were developed later. However, the theory is naturally very powerful, since it accommodates frequency and position simultaneously, and further it facilitates multi-resolution analysis - the analysis is then sensitive to scale which is advantageous since objects which are far from the camera appear smaller than those which are close. We shall find wavelets again, when processing images to find low-level features. Amongst applications of Gabor wavelets, we can find measurement of iris texture to give a very

powerful security system [Daugman93] and face feature extraction for automatic face recognition [Lades93]. Wavelets continue to develop [Daubechies90] and have found applications in image texture analysis [Laine93], in coding [daSilva96] and in image restoration [Banham96]. Unfortunately, the discrete wavelet transform is not shift invariant, though there are approaches aimed to remedy this (see for example [Donoho95]). As such, we shall not study it further and just note that there is an important class of transforms that combine spatial and spectral sensitivity, and it is likely that this importance will continue to grow.

### 2.7.3.2 Haar Wavelet

Though Fourier laid the basis for frequency decomposition, the original wavelet approach is now attributed to Alfred Haar's work in 1909. This uses a binary approach, rather than a continuous signal, and has led to fast methods for finding features in images [Oren97] (especially the object detection part of the Viola-Jones face detection approach [Viola01]). Essentially, the binary functions can be considered to form averages over sets of points, thereby giving means for compression and for feature detection. If we are to form a new vector (at level  $h+1$ ) by taking averages of pairs of elements (and retaining the integer representation) of the  $N$  points in the previous vector (at level  $h$  of the  $\log_2(N)$  levels) as,

$$\mathbf{p}_i^{h+1} = \frac{\mathbf{p}_{2^h i}^h + \mathbf{p}_{2^h i+1}^h}{2} \quad i \in 0 \dots \frac{N}{2} - 1; h \in 1, \dots, \log_2(N) \quad (2.51)$$

By way of example, consider a vector of points at level 0 as

$$\mathbf{p}^0 = [1 \ 3 \ 21 \ 19 \ 17 \ 19 \ 1 \ -1] \quad (2.52)$$

then the first element in the new vector becomes  $(1+3)/2 = 2$  and the next element is  $(21+19)/2 = 20$  and so on, so the next level is

$$\mathbf{p}^1 = [2 \ 20 \ 18 \ 0] \quad (2.53)$$

And is naturally half the number of points. If we also generate some detail, which is how we return to the original points, then we have a vector

$$\mathbf{d}^1 = [-1 \ 1 \ -1 \ 1] \quad (2.54)$$

and when each element of the detail  $\mathbf{d}^1$  is successively added and subtracted from the elements of  $\mathbf{p}^1$  as  $[\mathbf{p}_0^1 + \mathbf{d}_0^1 \ \mathbf{p}_0^1 - \mathbf{d}_0^1 \ \mathbf{p}_1^1 + \mathbf{d}_1^1 \ \mathbf{p}_1^1 - \mathbf{d}_1^1 \ \mathbf{p}_2^1 + \mathbf{d}_2^1 \ \mathbf{p}_2^1 - \mathbf{d}_2^1 \ \mathbf{p}_3^1 + \mathbf{d}_3^1 \ \mathbf{p}_3^1 - \mathbf{d}_3^1]$  by which we obtain

$$[2+(-1) \ 2-(-1) \ 20+1 \ 20-1 \ 18+(-1) \ 18-(-1) \ 0+1 \ 0-1]$$

which returns us to the original vector  $\mathbf{p}^0$  (Eqn. 2.52). If we continue to similarly form a series of decompositions (averages of adjacent points), together with the detail at each point, we generate

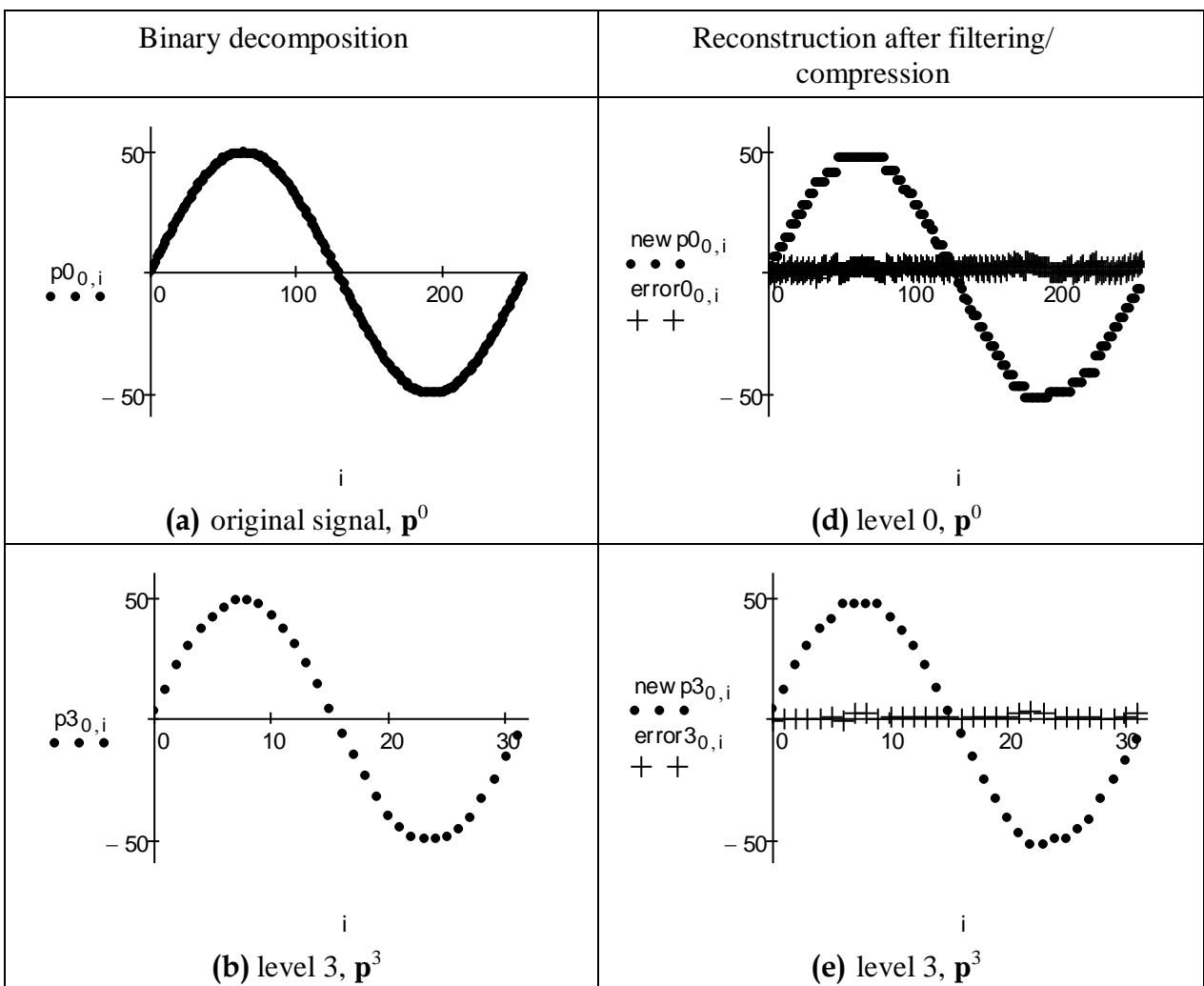
$$\mathbf{p}^2 = [11 \ 9]; \ \mathbf{d}^2 = [-9 \ 9] \quad (2.55)$$

$$\mathbf{p}^3 = [10]; \ \mathbf{d}^3 = [1] \quad (2.56)$$

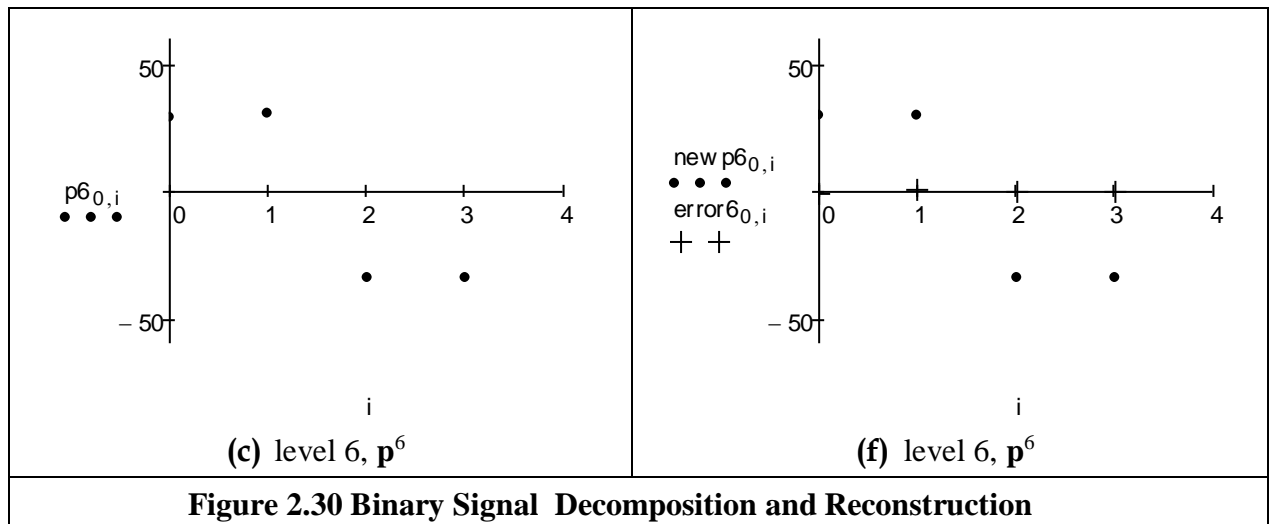
We can then store the image as a code

$$[\mathbf{p}^3 \ \mathbf{d}^3 \ \mathbf{d}^2 \ \mathbf{d}^1] = [10 \ 1 \ -9 \ 9 \ -1 \ 1 \ -1 \ 1] \quad (2.57)$$

The process is illustrated in Fig. 2.30 for a sinewave. Fig. 2.30(a) shows the original sinewave, (b) shows the decomposition to level 3, and it is a close but discrete representation whereas (c) shows the decomposition to level 6, which is very coarse. The original signal can be reconstructed from the final code, and this is without error. If the signal is reconstructed by filtering the detail to reduce the amount of stored data, the reconstruction of the original signal (d) at level 0 is quite close to the original signal, and the reconstruction at the other levels is similarly close as expected. The reconstruction error is also shown in (d) to (f). Components of the detail (of magnitude less than one) were removed, achieving a compression ratio of approximately 50%. Naturally, a Fourier transform would encode the signal better, as the Fourier transform is best suited to representing a sinewave. Like Fourier this discrete approach can encode the signal, we can also reconstruct the original signal (reverse the process), and shows how the signal can be represented at different scales, since there are less points in the higher levels.



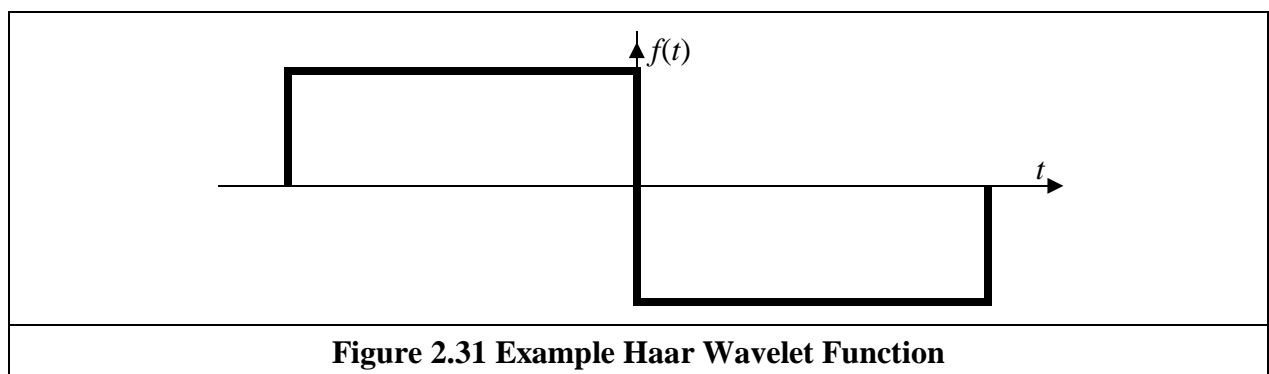




By Eqn. 2.57 this gives a set of numbers of the same size as the original data, and is an alternative representation from which we can reconstruct the original data. There are two important differences:

- i) we have an idea of scale by virtue of the successive averaging ( $p^1$  is similar in structure to  $p^0$ , but at a different scale) ;
- and ii) we can compress (or code) the image by removing the small numbers in the new representation (by setting them to zero, noting that there are efficient ways of encoding structures containing large numbers of zeros).

A process of successive averaging and differencing can be expressed as a function of the form in Fig. 2.31. This is a mother wavelet which can be applied at different scales, but retains the same shape at those scales. So we now have a binary decomposition rather than the sinewaves of the Fourier transform.

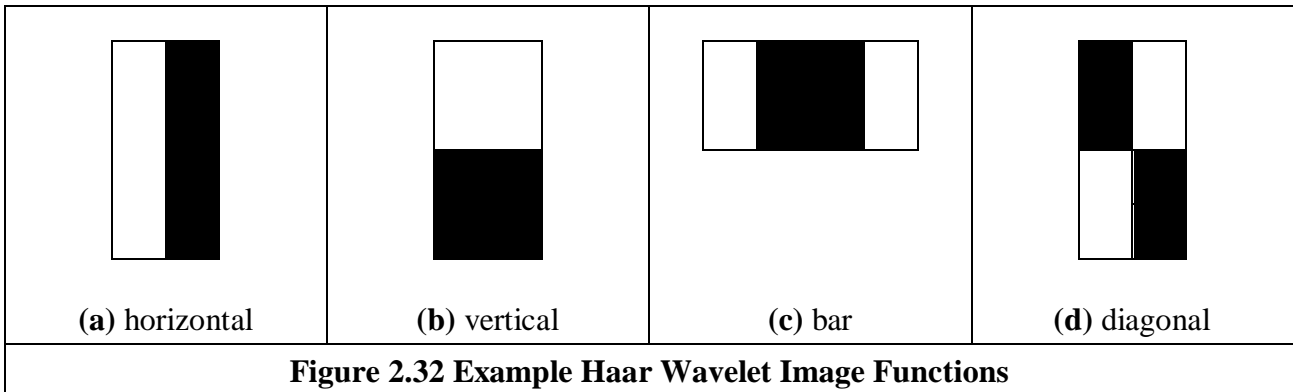


To detect objects, these wavelets need to be arranged in two dimensions. These can be arranged to provide for object detection, by selecting the two-dimensional arrangement of points. By defining a relationship which is a summation of the points in an image prior to a given point

$$sP_{x,y} = \sum_{x' < x, y' < y} P_{x',y'} \tag{2.58}$$

Then we can achieve wavelet type features which are derived by using these summations. Four of these wavelets are shown in Fig. 2.32. These are placed at selected positions in the image to which they are applied. There are white and black areas: the sum of the pixels under the white area(s) is

subtracted from of the sum of the pixels under the dark area(s), in a way similar to the earlier averaging operation in Eqn. 2.52. The first template, Fig. 2.32(a), will detect shapes which are brighter on one side than the other; the second (b) will detect shapes which are brighter in a vertical sense; the third will detect a dark object which has brighter areas on either side. There is a family of these arrangements, and that can apply at selected levels of scale. By collecting the analysis, we can determine objects whatever their position, size (objects further away will appear smaller) or rotation. We will dwell on these topics later and how we find and classify shapes. The point here is that we can achieve some form of binary decomposition in two dimensions, as opposed to the sine/cosine decomposition of the Gabor wavelet whilst retaining selectivity to scale and position (similar to the Gabor wavelet). This is also simpler, so the binary functions can be processed more quickly.



### 2.7.4 Other Transforms

Decomposing a signal into sinusoidal components was actually one of the first approaches to transform calculus, and this is why the Fourier transform is so important. The sinusoidal functions are actually called *basis functions*, the implicit assumption is that the basis functions map well to the signal components. As such, the Haar wavelets are binary basis functions. There is (theoretically) an infinite range of basis functions. Discrete signals can map better into collections of binary components rather than sinusoidal ones. These collections (or sequences) of binary data are called sequency components and form the basis of the *Walsh transform* [Walsh23], which is a global transform when compared with the Haar functions (like Fourier compared with Gabor). This has found wide application in the interpretation of digital signals, though it is less widely used in image processing (one disadvantage is the lack of shift invariance). The *Karhunen-Loève* transform [Karhunen47] [Loève48] (also called the *Hotelling* transform from which it was derived, or more popularly *Principal Component Analysis*) is a way of analysing (statistical) data to reduce it to those data which are informative, discarding those which are not.

## 2.8 Applications using Frequency Domain Properties

Filtering is a major use of Fourier transforms, particularly because we can understand an image, and how to process it, much better in the frequency domain. An analogy is the use of a graphic equaliser to control the way music sounds. In images, if we want to remove high-frequency information (like the hiss on sound) then we can filter, or remove, it by inspecting the Fourier transform. If we retain low-frequency components, we implement a *low-pass filter*. The low-pass filter describes the area in which we retain spectral components, the size of the area dictates the range of frequencies retained, and is known as the filter's bandwidth. If we retain components within a circular region centred on the d.c. component, and inverse Fourier transform the filtered transform then the resulting image will be blurred. Higher spatial frequencies exist at the sharp

edges of features, so removing them causes blurring. But the amount of fluctuation is reduced too; any high frequency noise will be removed in the filtered image.

The Matlab implementation of a low-pass filter which retains frequency components within a circle of specified radius is the function `low_filter`, given in Code 2.6(a). This operator assumes that the radius and centre co-ordinates of the circle are specified prior to its use. Points within the circle remain unaltered, whereas those outside the circle are set to zero, black. The Python version is given in Code 2.6(b). The function uses Fourier coefficients in the array `coeff`. This array is computed in Code 2.2. The value `cutFrequency` is used to select frequencies and should be set to between the maximum and minimum frequency value in the image. The code iterates over all frequencies and it computes the distance to the zero frequency. If the distance is lower than `cutFrequency` then the frequency is copied to the array `coeffLow`, otherwise it is copied to the `coeffHigh` array. That is, the code fills two arrays that contains the low and high frequencies in the image.

```
function output = low_filter(image,value)
%get dimensions
[rows,cols]=size(image);

%filter the transform
for x = 1:cols %address all columns
    for y = 1:rows %address all rows
        if ((y-(rows/2))^2)+((x-(cols/2))^2)-(value^2))>0
            output(y,x)=0; %discard components outside the circle
        else
            output(y,x)=image(y,x); %and keep the ones inside
        end
    end
end
end
```

(a) Matlab

```
for kw,kh in itertools.product(range(-maxFreqW, maxFreqW + 1), \
                               range(-maxFreqH, maxFreqH + 1)):
    IndexW, indexH = kw + maxFreqW, kh + maxFreqH

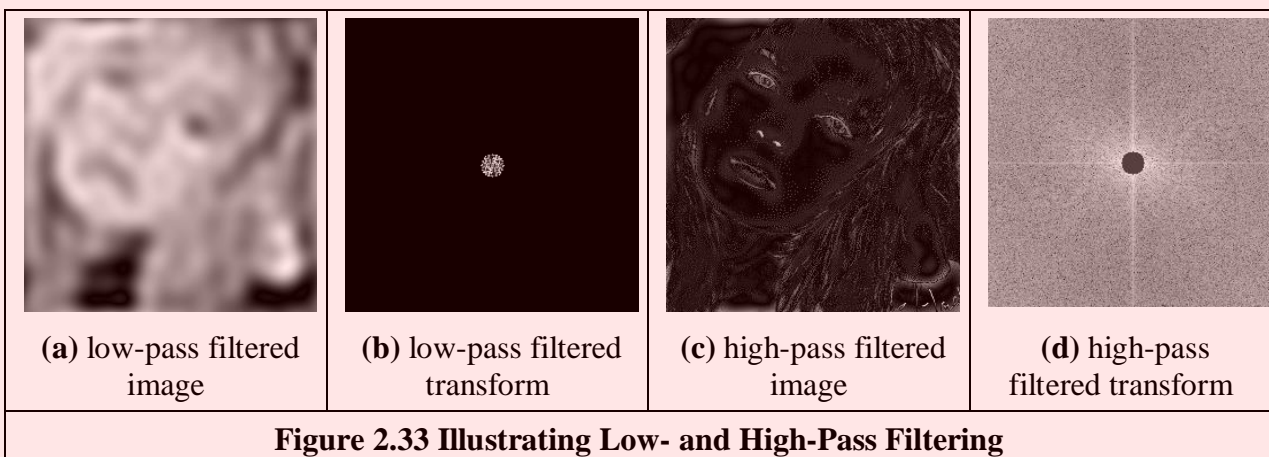
    if sqrt(kw * kw + kh * kh) < cutFrequency:
        coeffLow[indexH, IndexW][0] = coeff[indexH, IndexW][0]
        coeffLow[indexH, IndexW][1] = coeff[indexH, IndexW][1]
    else:
        coeffHigh[indexH, IndexW][0] = coeff[indexH, IndexW][0]
        coeffHigh[indexH, IndexW][1] = coeff[indexH, IndexW][1]
```

(b) Python

### Code 2.6 Implementing Low-Pass Filtering

When applied to an image we obtain a low-pass filtered version. In application to an image of a face, the low spatial frequencies are the ones which change slowly as reflected in the resulting, blurred image, Figure 2.33(a). The high frequency components have been removed as shown in the transform, Figure 2.33(b). The radius of the circle controls how much of the original image is retained. In this case, the radius is 10 pixels (and the image resolution is  $256 \times 256$ ). If a larger circle were to be used, more of the high frequency detail would be retained (and the image would look more like its original version); if the circle was very small, an even more blurred image would result, since only the lowest spatial frequencies would be retained. This differs from the earlier Gabor wavelet approach that allows for localised spatial frequency analysis. Here, the analysis is global: we are filtering the frequency across the whole image.

Alternatively, we can retain high frequency components and remove low frequency ones. This is a *high-pass filter*. If we remove components near the d.c. component and retain all the others, the result of applying the inverse Fourier transform to the filtered image will be to emphasise the features that were removed in low-pass filtering. This can lead to a popular application of the high-pass filter: to ‘crispen’ an image by emphasising its high frequency components. An implementation using a circular region merely requires selection of the set of points outside the circle, rather than inside as for the low-pass operator. The effect of high-pass filtering can be observed in Figure 2.33(c) which shows removal of the low frequency components: this emphasises the hair and the borders of a face’s features since these are where brightness varies rapidly. The retained components are those which were removed in low-pass filtering, as illustrated in the transform, Figure 2.33(d).



It is also possible to retain a specified range of frequencies. This is known as *band-pass filtering*. It can be implemented by retaining frequency components within an annulus centred on the d.c. component. The width of the annulus represents the bandwidth of the band-pass filter.

This leads to digital signal processing theory. There are many considerations to be made in the way you select, and the manner in which frequency components are retained or excluded. This is beyond a text on Computer Vision. For further study in this area, Rabiner and Gold [Rabiner75], or Oppenheim and Schaffer [Oppenheim09], although published (in their original form) a long time ago now, remain as popular introductions to Digital Signal Processing theory and applications.

It is actually possible to recognise the object within the low-pass filtered image. Intuitively, this implies that we could just store the frequency components selected from the transform data, rather than all the image points. In this manner a fraction of the information would be stored, and still provide a recognizable image, albeit slightly blurred. This concerns *image coding* which is a popular target for image processing techniques, for further information see [Clarke85] or a newer text, like [Woods11] or [Sayood17]. Note that the JPEG coding approach uses frequency domain decomposition, and is arguably the most ubiquitous image coding technique used today.

## 2.9 Further Reading

We shall meet the frequency domain throughout this book, since it allows for an alternative interpretation of operation, in the frequency domain as opposed to the time domain. This will occur in low- and high-level feature extraction, and in shape description. Further, it actually allow for some of the operations we shall cover. Further, because of the availability of the FFT, it is also used to speed up algorithms.

Given these advantages, it is well worth looking more deeply. Mark's copy of Fourier's original book has a review "Fourier's treatise is one of the very few scientific books which can never be rendered antiquated by the progress of science" – penned by James Clerk Maxwell no less. For introductory study, there is *Who is Fourier* [Lex12] which offers a lighthearted and completely digestible overview of the Fourier transform, it's simply excellent for a starter view of the topic. For further study (and entertaining study too!) of the Fourier transform, try *The Fourier Transform and its Applications* by R. N. Bracewell [Bracewell86], or a newer text [Nussbaumer12]. A number of the standard image processing texts include much coverage of transform calculus, such as Jain [Jain89], Gonzalez and Wintz [Gonzalez17], and Pratt [Pratt13]. There is a relatively new text concentrating on image processing using Python [Chityala15]. For more coverage of the DCT try Jain [Jain89]; for an excellent coverage of the Walsh transform try Beauchamp's superb text [Beauchamp75]. On compressed sensing, [Eldar12] is worth a read. For wavelets, try the book by Wornell that introduces wavelets from a signal processing standpoint [Wornell96], there's Mallat's classic text [Mallat08] or a new text that includes images [Broughton18]. For general signal processing theory there are introductory texts (see for example Meade and Dillon [Meade86], or Ifeachor's excellent book [Ifeachor02]), for more complete coverage try Rabiner and Gold [Rabiner75] or Oppenheim and Schaffer [Oppenheim09] (as mentioned earlier). Finally, on the implementation side of the FFT (and for many other signal processing algorithms) *Numerical Recipes in C* [Press07] is an excellent book. It is extremely readable, full of practical detail – well worth a look. Numerical Recipes is on the web too, together with other signal processing sites, as listed in Table 1.4.

## 2.10 Chapter 2 References

- [Ahmed74] Ahmed, N., Natarajan, T. and Rao, K. R., Discrete Cosine Transform, *IEEE Trans. on Computers*, pp 90-93, 1974
- [Banham96] Banham, M. R., and Katsaggelos, K., Spatially Adaptive Wavelet-Based Multiscale Image Restoration, *IEEE Trans. on Image Processing*, **5**(4), pp 619-634 , 1996
- [Beauchamp75] Beauchamp, K. G., *Walsh Functions and Their Applications*, Academic Press, London UK, 1975
- [Bracewell84] Bracewell, R. N., The Fast Hartley Transform, *Proc. IEEE*, **72**(8), pp 1010-1018, 1984
- [Bracewell83] Bracewell, R. N., The Discrete Hartley Transform, *J. Opt. Soc. Am.*, **73**(12), pp 1832-1835, 1984
- [Bracewell86] Bracewell, R. N., *The Fourier Transform and its Applications*, Revised 2<sup>nd</sup> Edition, McGraw-Hill, Book Co., Singapore, 1986
- [Broughton18] Broughton, S. A., Bryan, K., *Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing*, Wiley, Chichester UK, 2<sup>nd</sup> Edition 2018
- [Chityala15] Chityala, R., and Pudipeddi, S., *Image Processing and Acquisition using Python*, CRC Press, Boca Raton FL USA, 2015
- [Clarke85] Clarke, R. J., *Transform Coding of Images*, Addison Wesley, Reading MA USA, 1985
- [daSilva96] da Silva, E. A. B., and Ghanbari, M., On the Performance of Linear Phase Wavelet Transforms in Low Bit-Rate Image Coding, *IEEE Trans. on Image Processing*, **5**(5), pp 689-704, 1996
- [Daubechies90] Daubechies, I., The Wavelet Transform, Time Frequency Localisation and Signal Analysis, *IEEE Trans. on Information Theory*, **36**(5), pp 961-1004, 1990

- [Daugman88] Daugman, J. G., Complete Discrete 2D Gabor Transforms by Neural Networks for Image Analysis and Compression, *IEEE Trans. on Acoustics, Speech and Signal Processing*, **36**(7), pp 1169-1179, 1988
- [Daugman93] Daugman, J. G., High Confidence Visual Recognition of Persons by a Test of Statistical Independence, *IEEE Trans. on PAMI*, **15**(11), pp1148-1161, 1993
- [Donoho95] Donoho, D. L., Denoising by Soft Thresholding, *IEEE Trans. on Information Theory*, **41**(3), pp 613-627, 1995
- [Donoho06] Donoho, D. L., Compressed Sensing, *IEEE Trans. on Information Theory*, **52**(4), pp 1289–1306, 2006
- [Eldar12] Eldar, Y. C., and Kutyniok, G., Editors, *Compressed Sensing: Theory and Applications*, Cambridge University Press, Cambridge UK, 2012
- [Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4<sup>th</sup> Edition, Pearson Education, 2017
- [Hartley42] Hartley, R. L. V., A More Symmetrical Fourier Analysis Applied to Transmission Problems, *Proc. IRE*, **144**, pp 144-150, 1942
- [Ifeachor02] Ifeachor, E. C., and Jervis, B. W., *Digital Signal Processing*, Prentice Hall, Hemel Hempstead UK, 2<sup>nd</sup> Edition 2002
- [Jain89] Jain A. K., *Fundamentals of Computer Vision*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 1989
- [Karhunen47] Karhunen, K., Über Lineare Methoden in der Wahrscheinlich-Keitsrechnung, *Ann. Acad. Sci. Fennicae*, Ser A.I.37, 1947 (Translation in I. Selin, On Linear Methods in Probability Theory, Doc. T-131, The RAND Corp., Santa Monica CA, 1960.)
- [Lades93] Lades, M., Vorbruggen, J. C., Buhmann, J., and Lange, J., Madsburg, C. V. D., Wurtz, R. P., and Konen, W., Distortion Invariant Object Recognition in the Dynamic Link Architecture, *IEEE Trans. on Computers*, **42**, pp 300-311, 1993
- [Laine93] Laine, A., and Fan, J., Texture Classification by Wavelet Packet Signatures, *IEEE Trans. on PAMI*, **15**, pp 1186-1191, 1993
- [Lex12] Lex, T. C. O. L. T. (!), *Who is Fourier, a Mathematical Adventure*, Language Research Foundation, Boston MA USA, 2<sup>nd</sup> Edition 2012
- [Loève48] Loève, M., Fonctions Aléatoires de Seconde Ordre, in: P. Levy, Ed., *Processus Stochastiques et Mouvement Brownien*, Hermann, Paris 1948
- [Mallat08] Mallat, S., *A Wavelet Tour of Signal Processing*, 3<sup>rd</sup> Edition, Academic Press, Burlington MA USA, 2008
- [Meade86] Meade, M. L. and Dillon, C. R., *Signals and Systems, Models and Behaviour*, Van Nostrand Reinhold (UK) Co. Ltd., Wokingham UK, 1986
- [Nussbaumer12] Nussbaumer, H. J., Fast Fourier Transform and Convolution Algorithms, Springer Science and Business Media, 2<sup>nd</sup> Ed., Berlin Germany, 2012
- [Oppenheim09] Oppenheim, A. V., Schafer, R. W., and Buck, J. R. *Discrete-Time Signal Processing*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 3<sup>rd</sup> Edition, 2009
- [Oren97] Oren, M., Papageorgiou, C., Sinha, P., Osuna, E., and Poggio, T., Pedestrian Detection Using Wavelet Templates, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, pp193-199, 1997
- [Pratt13] Pratt, W. K., *Introduction to Digital Image Processing*, CRC Press, Boca Raton FL USA, 2013
- [Press07] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes 3<sup>rd</sup> Edition: The Art of Scientific Computing*, Cambridge University Press, 3<sup>rd</sup> Edition, 2007

- [Rabiner75] Rabiner, L. R. and Gold, B., *Theory and Application of Digital Signal Processing*, Prentice Hall Inc., Englewood Cliffs NJ USA, 1975
- [Sayood17] Sayood, K., *Introduction to Data Compression*, Morgan Kaufmann, Cambridge MA USA, 5<sup>th</sup> Edition 2017
- [Unser00] Unser, M., Sampling - 50 Years after Shannon, *Proceedings of the IEEE*, **88**(4), pp 569-587, 2000
- [Viola01] Viola, P., and Jones, M., Rapid Object Detection using a Boosted Cascade of Simple Features, *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)*, **1**, pp.511-519, 2001
- [Walsh23] Walsh, J. L., A Closed Set of Normal Orthogonal Functions, *Am. J. Math.*, **45**(1), pp 5-24, 1923
- [Woods11] Woods, J. W., *Multidimensional Signal, Image, and Video Processing and Coding*, Academic Press, Wakham MA USA, 2<sup>nd</sup> Edition 2011
- [Wornell96] Wornell, G. W., *Signal Processing with Fractals, a Wavelet-Based Approach*, Prentice Hall Inc., Upper Saddle River NJ USA, 1996

## 3 Image Processing

### 3.1 Overview

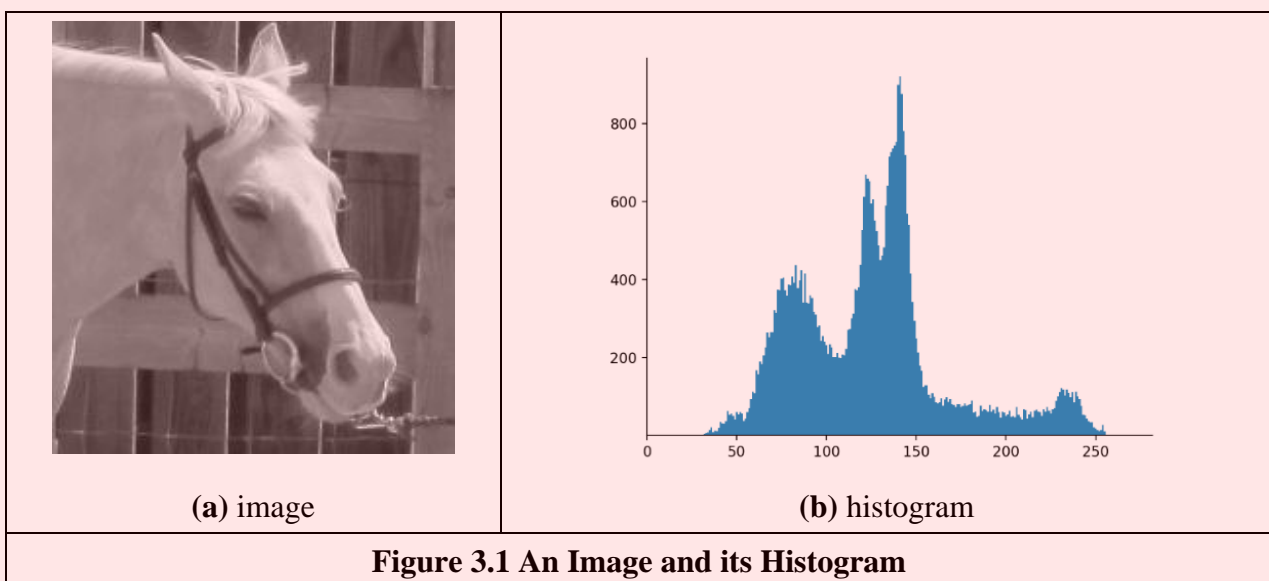
We shall now start to process digital images. First, we shall describe the brightness variation in an image using its histogram. We shall then look at operations which manipulate the image so as to change the histogram, and at processes that shift and scale the result (making the image brighter or dimmer, in different ways). We shall also consider thresholding techniques that turn an image from grey level into binary. These are called single point operations. After, we shall move to group operations where the group is those points found inside a template. Some of the most common operations on the groups of points are statistical, providing images where each point is the result of, say, averaging the neighbourhood of each point in the original image. We shall see how the statistical operations can reduce noise in the image, which is of benefit to the feature extraction techniques to be considered later. As such, these basic operations are usually for pre-processing for later feature extraction or to improve display quality. Finally, morphological operators process an image according to shape, starting with binary and moving to grey level operations.

Main topic	Sub topics	Main points
Image Description	Portray variation in image brightness content as a graph/histogram.	<i>Histograms, image contrast.</i>
Point Operations	Calculate new image points as a function of the point at the same place in the original image. The functions can be mathematical, or can be computed from the image itself and will change the image's histogram. Finally, thresholding turns an image from grey level to a binary (black and white) representation.	<i>Histogram manipulation; intensity mapping: addition, inversion, scaling, logarithm, exponent. Intensity normalisation; histogram equalisation. Thresholding and optimal thresholding.</i>
Group Operations	Calculate new image points as a function of the neighbourhood of the point at the same place in the original image. The functions can be statistical including: mean (average); median and mode. Advanced filtering techniques, including feature preservation.	<i>Template convolution (inc. frequency domain implementation). Statistical operators: direct averaging, median filter, and mode filter. Non-local means, anisotropic diffusion, and bilateral filter for image smoothing. Other operators: force field and image ray transforms.</i>
Image morphology and operators	Morphological operators that process an image according to shape, starting with binary and moving to grey level operations.	<i>Mathematical morphology: hit or miss transform, erosion, dilation (inc. grey level operators) and Minkowski operators.</i>
<b>Table 3.1 Overview of Chapter 3</b>		



## 3.2 Histograms

The intensity *histogram* shows how individual brightness levels are occupied in an image; the *image contrast* is measured by the range of brightness levels. The histogram plots the number of pixels with a particular brightness level against the brightness level. For 8-bit pixels, the brightness ranges from zero (black) to 255 (white). Figure 3.1 shows an image and its histogram. The histogram, Figure 3.1(b), shows that not all the grey levels are used and the lowest and highest intensity levels are far apart, reflecting good contrast. Note that the image contains many light grey pixels that produce the wide lower peak in the histogram. If the image was darker, overall, the histogram would be concentrated towards black. If the image was brighter, but with lower contrast, then the histogram would be thinner and concentrated near the whiter brightness levels.



This histogram shows us that we have not used all available grey levels. Accordingly, we can stretch the image to use them all, and the image would become clearer. This is essentially cosmetic attention to make the image's appearance better. Making the appearance better, especially in view of later processing, is the focus of many basic image processing operations, as will be covered in this Chapter. The histogram can also reveal if there is much noise in the image, if the ideal histogram is known. We might want to remove this noise, not only to improve the appearance of the image, but to ease the task of (and to present the target better for) later feature extraction techniques. This Chapter concerns these basic operations which can improve the appearance and quality of images.

## 3.3 Point Operators

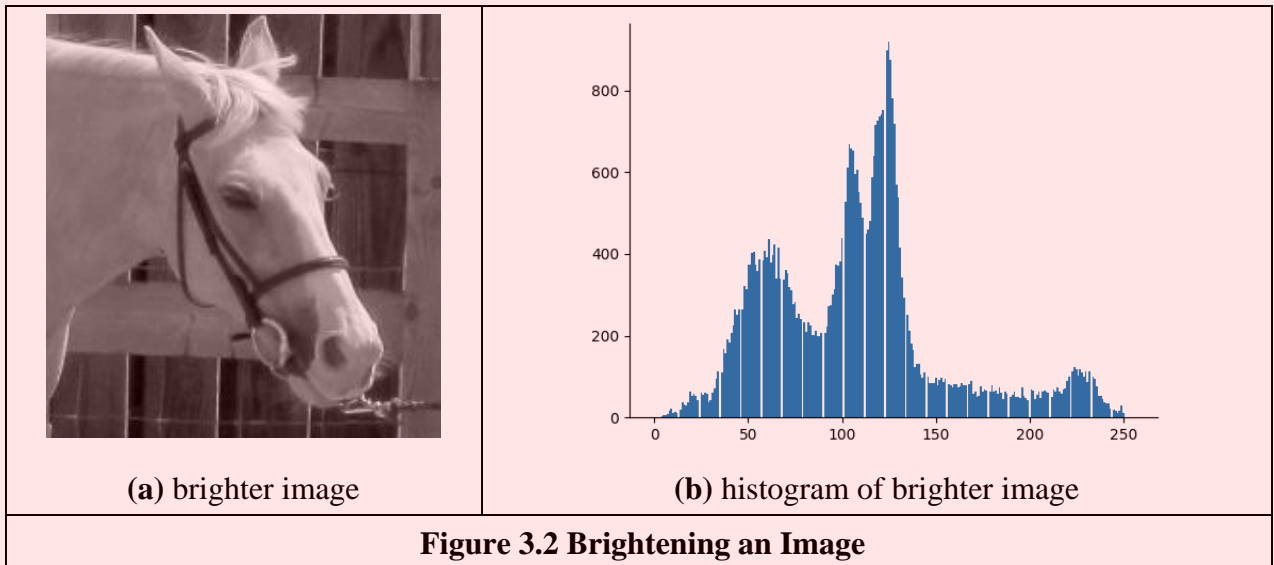
### 3.3.1 Basic Point Operations

The most basic operations in image processing are *point operations* where each pixel value is replaced with a new value obtained from the old one. If we want to increase the brightness to stretch the contrast we can simply multiply all pixel values by a scalar, say by 2 to double the range. Conversely, to reduce the contrast (though this is not usual) we can divide all point values by a

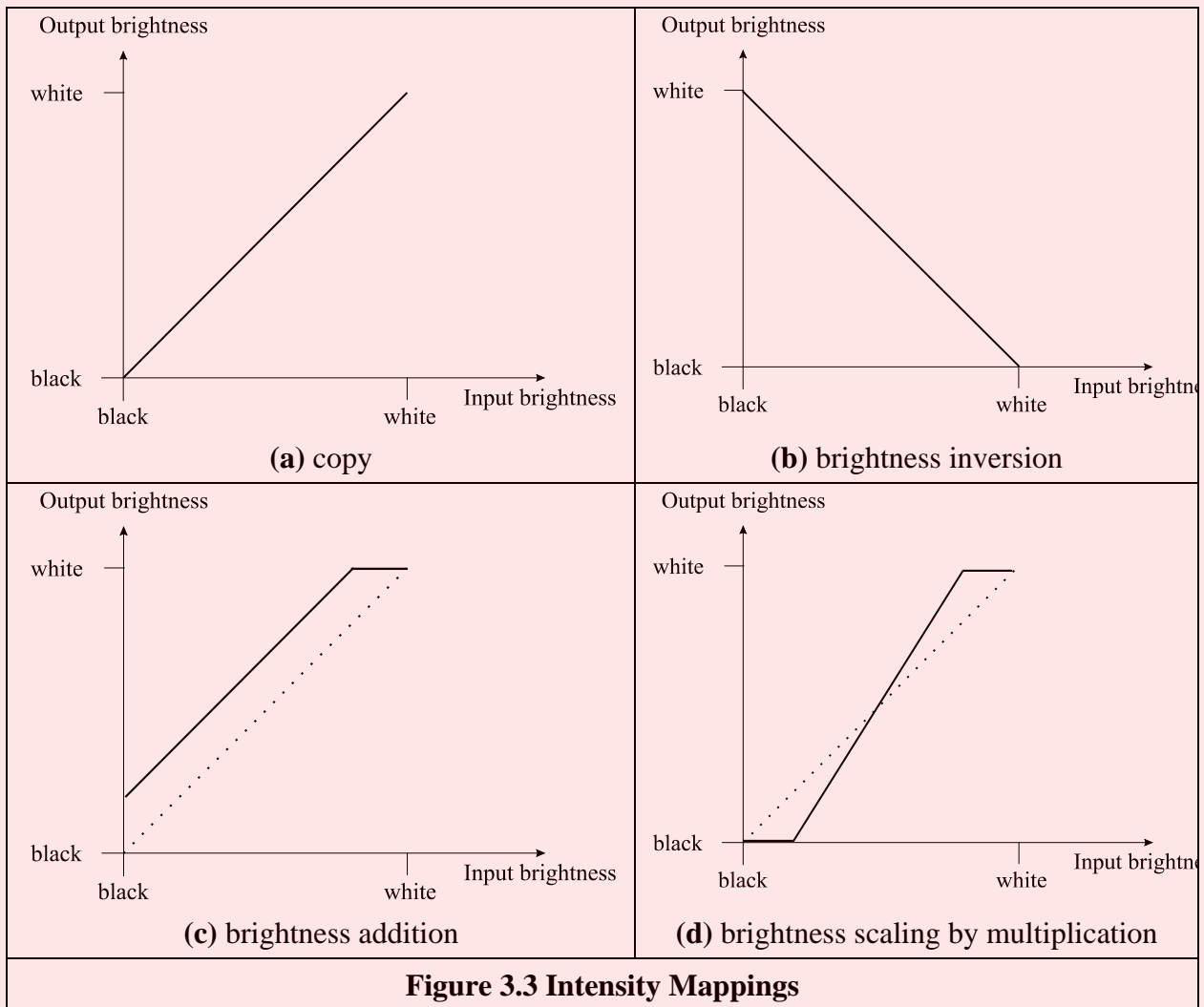
scalar. If the overall brightness is controlled by a *level*,  $l$ , (e.g. the brightness of global light) and the range is controlled by a *gain*,  $k$ , the brightness of the points in a new picture,  $\mathbf{N}$ , can be related to the brightness in old picture,  $\mathbf{O}$ , by:

$$\mathbf{N}_{x,y} = k \times \mathbf{O}_{x,y} + l \quad \forall x, y \in 1, N \quad (3.1)$$

This is a point operator that replaces the brightness at points in the picture according to a linear brightness relation. The level controls overall brightness and is the minimum value of the output picture. The gain controls the contrast, or range, and if the gain is greater than unity, the output range will be increased, this process is illustrated in Figure 3.2. So the image, processed by  $k = 1.1$  and  $l = -10$  will become brighter, Figure 3.2(a), and with better contrast, though in this case the brighter points are mostly set near to white (255). These factors can be seen in its histogram, Figure 3.2(b).



The basis of the implementation of point operators was given earlier, for inversion in Code 1.2. The stretching process can be displayed as a mapping between the input and output ranges, according to the specified relationship, as in Figure 3.3. Figure 3.3(a) is a mapping where the output is a direct copy of the input (this relationship is the dotted line in Figures 3.3(c) and (d)); Figure 3.3(b) is the mapping for *brightness inversion* where dark parts in an image become bright and vice versa. Figure 3.3(c) is the mapping for *addition* and Figure 3.3(d) is the mapping for *multiplication* (or *division*, if the slope was less than that of the input). In these mappings, if the mapping produces values that are smaller than the expected minimum (say negative when zero represents black), or larger than a specified maximum, then a *clipping* process can be used to set the output values to a chosen level. For example, if the relationship between input and output aims to produce output points with intensity values greater than 255, as used for white, the output value can be set to white for these points, as it is in Figure 3.2.



Finally, rather than simple multiplication we can use arithmetic functions such as the logarithm to reduce the range or the exponent to increase it. This can be used, say, to equalise the response of a camera, or to compress the range of displayed brightness levels. If the camera has a known exponential performance, and outputs a value for brightness which is proportional to the exponential of the brightness of the corresponding point in the scene of view, the application of a *logarithmic point operator* will restore the original range of brightness levels. The implementation of point operators for arithmetic functions is illustrated in Code 3.1. This code simply computes the logarithm or exponential value of the value at each pixel position.

```

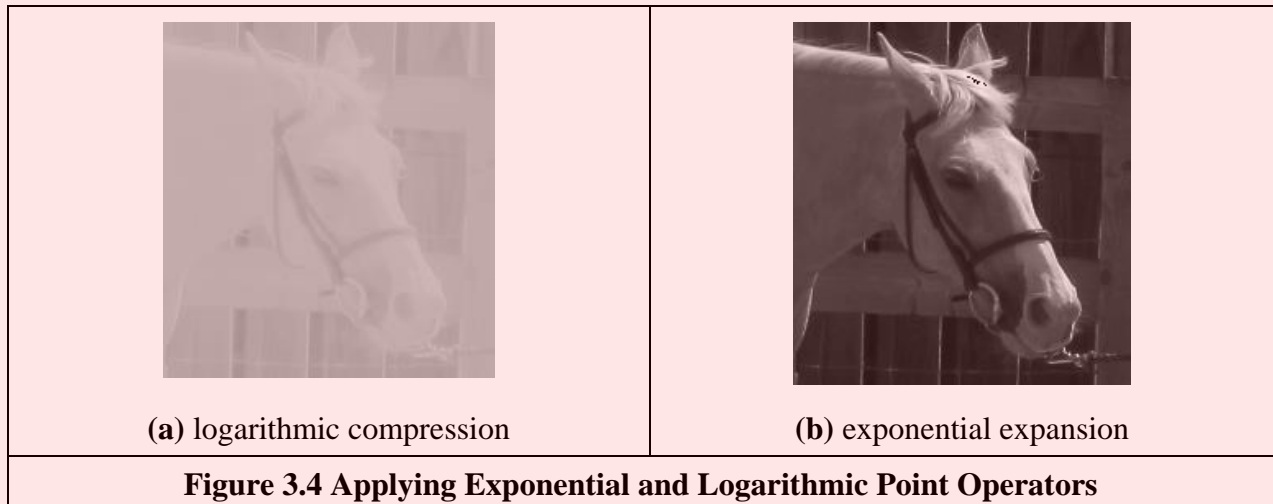
for x,y in itertools.product(range(0, width), range(0, height)):
    # Set the pixels in the Logarithmic
    outputLogarithmicImage[y,x] = 20 * log(inputImage[y,x] * 100.0)

    # Set the pixels in the Exponential image
    outputExponentialImage[y,x] = 20 * exp(inputImage[y,x] / 100.0)
    
```

**Code 3.1 Point Operations**

The effect of replacing brightness by a scaled version of its natural logarithm (implemented as  $N_{x,y} = 20\ln(100O_{x,y})$ ) is shown in Figure 3.4(a); the effect of a scaled version of the exponent (implemented as  $N_{x,y} = 20\exp(O_{x,y}/100)$ ) is shown in Figure 3.4(b). The scaling factors were chosen to ensure that the resulting image can be displayed since the logarithm or exponent greatly reduce or magnify, pixel values, respectively. This can be seen in the results: Figure 3.4(a) is dark with a small range of brightness levels whereas Figure 3.4(b) is much brighter, with greater contrast.

Naturally, application of the logarithmic point operator will change any multiplicative changes in brightness to become additive. As such, the logarithmic operator can find application in reducing the effects of multiplicative intensity change. The logarithm operator is often used to compress Fourier transforms for display purposes. This is because the d.c. component can be very large, or the contrast too large, to allow the other points to be seen.



In hardware, point operators can be implemented using *look-up-tables* (LUTs). LUTs give an output that is programmed, and stored, in a table entry that corresponds to a particular input value. If the brightness response of the camera is known, it is possible to pre-program an LUT to make the camera response equivalent to a uniform or flat response across the range of brightness levels (in software, this can be implemented as an array or by using a map associative container).

### 3.3.2 Histogram Normalisation

Popular techniques to stretch the range of intensities include *histogram (intensity) normalisation*. Here, the original histogram is stretched, and shifted, to cover all the 256 available levels. If the original histogram of an old picture  $\mathbf{O}$  starts at  $\mathbf{O}_{min}$  and extends up to  $\mathbf{O}_{max}$  brightness levels, then we can scale up the image so that the pixels in the new picture  $\mathbf{N}$  lie between a minimum output level  $\mathbf{N}_{min}$  and a maximum level  $\mathbf{N}_{max}$ , simply by scaling up the input intensity levels according to:

$$\mathbf{N}_{x,y} = \frac{\mathbf{N}_{max} - \mathbf{N}_{min}}{\mathbf{O}_{max} - \mathbf{O}_{min}} \times (\mathbf{O}_{x,y} - \mathbf{O}_{min}) + \mathbf{N}_{min} \quad \forall x, y \in 1, N \quad (3.2)$$

Code 3.2 gives an implementation of intensity normalisation. The code uses an output ranging from  $\mathbf{N}_{min} = 0$  to  $\mathbf{N}_{max} = 255$  that is the maximum range for images that use a byte per pixel. This is scaled by the input range that is determined from the maximum and minimum values are returned by the `imageMaxMin` function. Each point in the picture is then scaled as in Equation 3.2. Note that Matlab's `imagesc` function appears to effect normalisation.

```
# Maximum and range
maxVal, miniVal = imageMaxMin(inputImage)
brightRange = float(maxVal - miniVal)

# Set the pixels in the output image
for x,y in itertools.product(range(0, width), range(0, height)):

    # Normalize the pixel value according to the range
```

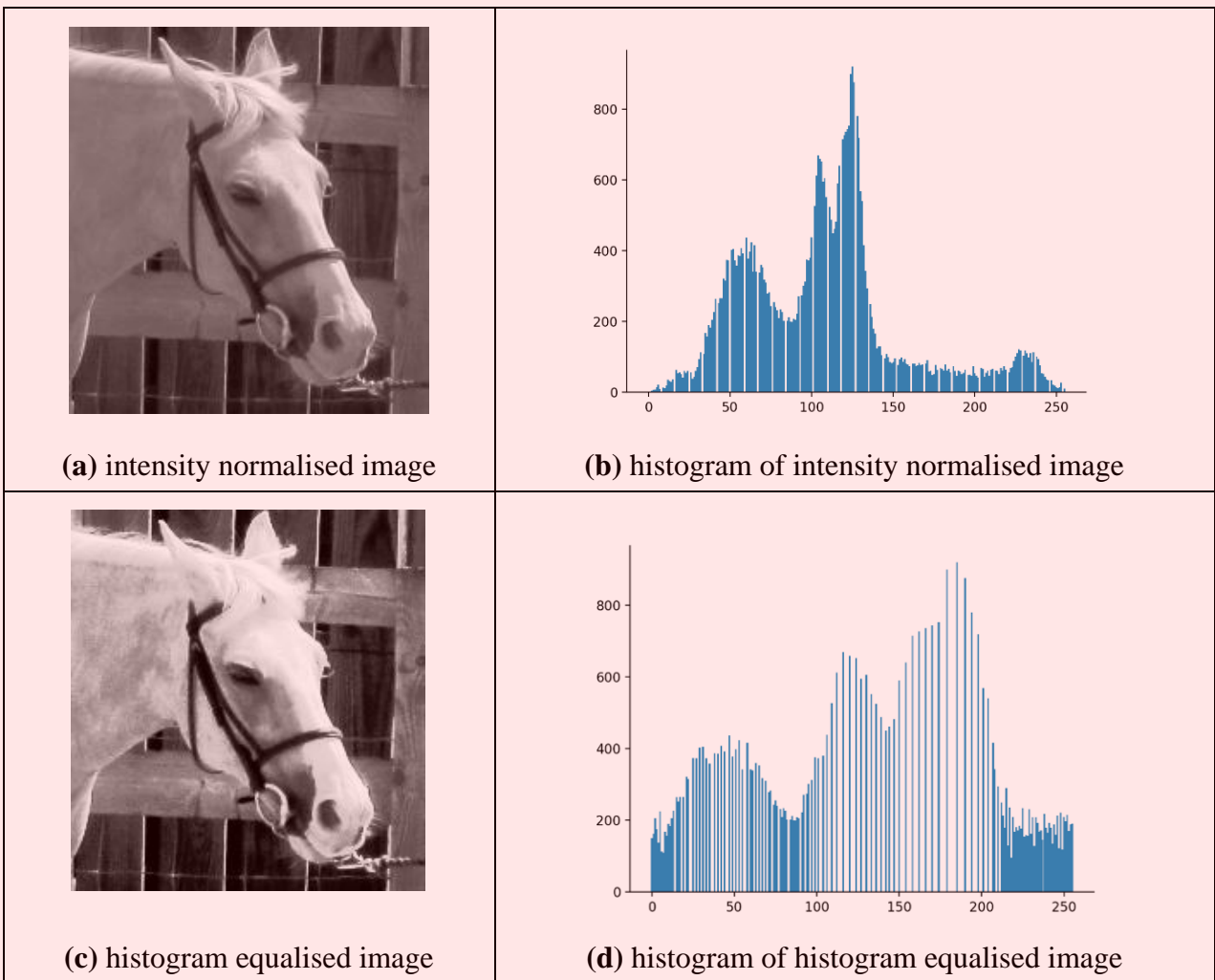
```
outputNormalizedImage[y,x] = ((inputImage[y,x] - miniVal) * 255.0 / brightRange)
```

**Code 3.2 Intensity Normalisation**

The normalisation process is illustrated in Figure 3.5, and can be compared with the original image and histogram in Figure 3.1. An intensity-normalised version of the image is shown in Figure 3.5(a) which now has better contrast and appears better to human vision. Its histogram, Figure 3.5(b), shows that the intensity now ranges across all available levels (there is actually one black pixel!).

**3.3.3 Histogram Equalisation**

*Histogram equalisation* is a non-linear process aimed to highlight image brightness in a way particularly suited to human visual analysis. Histogram equalisation aims to change a picture in such a way as to produce a picture with a flatter histogram, where all levels are equiprobable. In order to develop the operator, we can first inspect the histograms. For a range of  $M$  levels then the histogram plots the points per level against level. For the input (old) and the output (new) image, the number of points per level is denoted as  $O(l)$  and  $N(l)$  (for  $0 < l < M$ ), respectively. For square images, there are  $N^2$  points in the input and the output image, so the sum of points per level in each should be equal:



**Figure 3.5 Illustrating Intensity Normalisation and Histogram Equalisation**

$$\sum_{l=0}^M \mathbf{O}(l) = \sum_{l=0}^M \mathbf{N}(l) \quad (3.3)$$

Also, this should be the same for an arbitrarily chosen level  $p$ , since we are aiming for an output picture with a uniformly flat histogram. So the cumulative histogram up to level  $p$  should be transformed to cover up to the level  $q$  in the new histogram:

$$\sum_{l=0}^p \mathbf{O}(l) = \sum_{l=0}^q \mathbf{N}(l) \quad (3.4)$$

Since the output histogram is uniformly flat, the cumulative histogram up to level  $p$  should be a fraction of the overall sum. So the number of points per level in the output picture is the ratio of the number of points to the range of levels in the output image:

$$\mathbf{N}(l) = \frac{N^2}{\mathbf{N}_{max} - \mathbf{N}_{min}} \quad (3.5)$$

So the cumulative histogram of the output picture is:

$$\sum_{l=0}^q \mathbf{N}(l) = q \times \frac{N^2}{\mathbf{N}_{max} - \mathbf{N}_{min}} \quad (3.6)$$

By Equation 3.4 this is equal to the cumulative histogram of the input image, so that:

$$q \times \frac{N^2}{\mathbf{N}_{max} - \mathbf{N}_{min}} = \sum_{l=0}^p \mathbf{O}(l) \quad (3.7)$$

This gives a mapping for the output pixels at level  $q$ , from the input pixels at level  $p$  as:

$$q = \frac{\mathbf{N}_{max} - \mathbf{N}_{min}}{N^2} \times \sum_{l=0}^p \mathbf{O}(l) \quad (3.8)$$

This gives a mapping function that provides an output image that has an approximately flat histogram. The mapping function is given by phrasing Equation 3.8 as an equalising function ( $E$ ) of the level ( $q$ ) and the image ( $\mathbf{O}$ ) as

$$E(q, \mathbf{O}) = \frac{\mathbf{N}_{max} - \mathbf{N}_{min}}{N^2} \times \sum_{l=0}^p \mathbf{O}(l) \quad (3.9)$$

The output image is then

$$\mathbf{N}_{x,y} = E(\mathbf{O}_{x,y}, \mathbf{O}) \quad (3.10)$$

```
function equalised = equalise(image)
%get dimensions
[rows,cols]=size(image);
%specify range of levels
range=255;
%and the number of points
number=cols*rows;

%initialise the image histogram
hist(1:256)=0;

%work out the histogram
for x = 1:cols %address all columns
    for y = 1:rows %address all rows
        hist(image(y,x)+1)=hist(image(y,x)+1)+1;
    end
end;

%evaluate the cumulative histogram
sum=0;
for i=1:256
```

```

sum=sum+hist(i);
cumhist(i)=floor(sum*range/number); %Eq. 3.9
end

%map using the cumulative histogram
for x = 1:cols %address all columns
    for y = 1:rows %address all rows
        equalised(y,x)=cumhist(image(y,x)); %Eq 3.10
    end
end
end

```

### Code 3.3 Histogram Equalisation

The result of equalising an image is shown in Figures 3.5(c) and (d). The intensity equalised image, Figure 3.5(c) has much better defined features than in the original version (Figure 3.1). The histogram, Figure 3.5(d), reveals the non-linear mapping process whereby white and black are not assigned equal weight, as they were in intensity normalisation. Accordingly, more pixels are mapped into the darker region and the brighter intensities become better spread, consistent with the aims of histogram equalisation.

Its performance can be very convincing since it is well mapped to the properties of human vision. If a linear brightness transformation is applied to the original image then the equalised histogram will be the same. If we replace pixel values with ones computed according to Equation 3.1, the result of histogram equalisation will not change. An alternative interpretation is that if we equalise images (prior to further processing) then we need not worry about any brightness transformation in the original image. This is to be expected, since the linear operation of the brightness change in Equation 3.2 does not change the overall shape of the histogram, only its size and position. However, noise in the image acquisition process will affect the shape of the original histogram, and hence the equalised version. So the equalised histogram of a picture will not be the same as the equalised histogram of a picture with some noise added to it. You cannot avoid noise in electrical systems, however well you design a system to reduce its effect. Accordingly, histogram equalisation finds little use in generic image processing systems, except for display, though it can be potent in specialised applications. For these reasons, intensity normalisation is often preferred when a picture's histogram requires manipulation.

In implementation, the function `equalise` in Code 3.3, we shall use an output range where  $N_{min} = 0$  and  $N_{max} = 255$ . The implementation first determines the cumulative histogram for each level of the brightness histogram. This is then used as a look up table for the new output brightness at that level. The look up table is used to speed implementation of Equation 3.9, since it can be precomputed from the image to be equalised.

An alternative argument also against the use of histogram equalisation is that it is a non-linear process and is irreversible. We cannot return to the original picture after equalisation, and we cannot separate the histogram of an unwanted picture. On the other hand, intensity normalisation is a linear process and we can return to the original image, should we need to, or separate pictures - if required. Note that there have been extensions to histogram equalisation, and *adaptive histogram equalisation* with some extensions [Pizer87] has proved particularly enduring.

#### 3.3.4 Thresholding

The last point operator of major interest is called *thresholding*. This operator selects pixels which have a particular value, or are within a specified range. It can be used to find objects within a picture if their brightness level (or range) is known. This implies that the object's brightness must be known as well. There are two main forms: uniform and adaptive thresholding. In *uniform thresholding*, pixels above a specified level are set to white, those below the specified level are set to black.

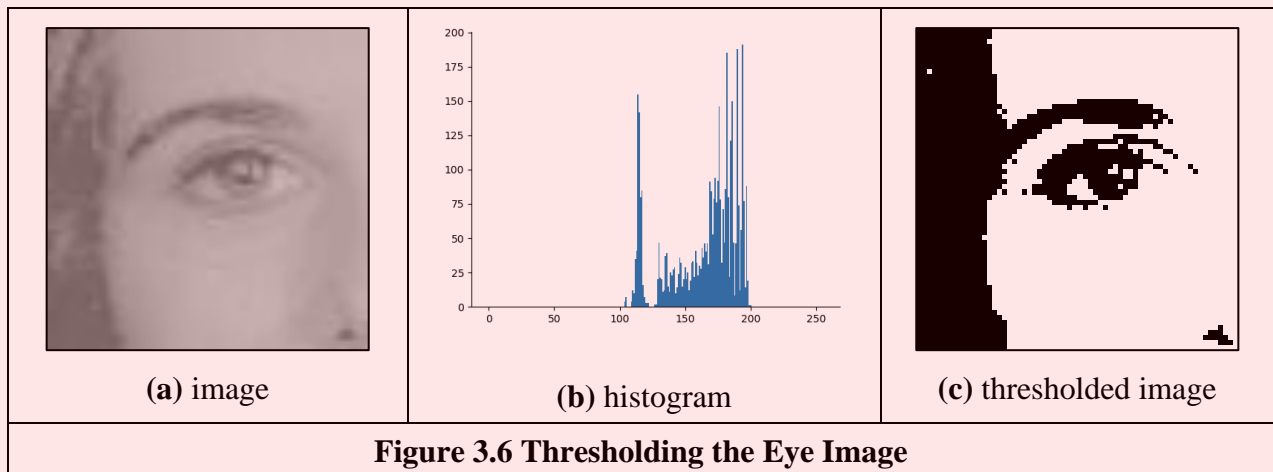
$$N_{x,y} = \begin{cases} 255 & \text{if } O_{x,y} > \text{threshold} \\ 0 & \text{otherwise} \end{cases} \quad (3.11)$$

As shown in Code 3.4 the implementation of thresholding sets the value of the output image by an `if` condition according to the threshold parameter.

```
for x,y in itertools.product(range(0, width), range(0, height)):
    if inputImage[y,x] > threshold:
        outputImage[y,x] = 255
    else:
        outputImage[y,x] = 0
```

**Code 3.4 Image Thresholding**

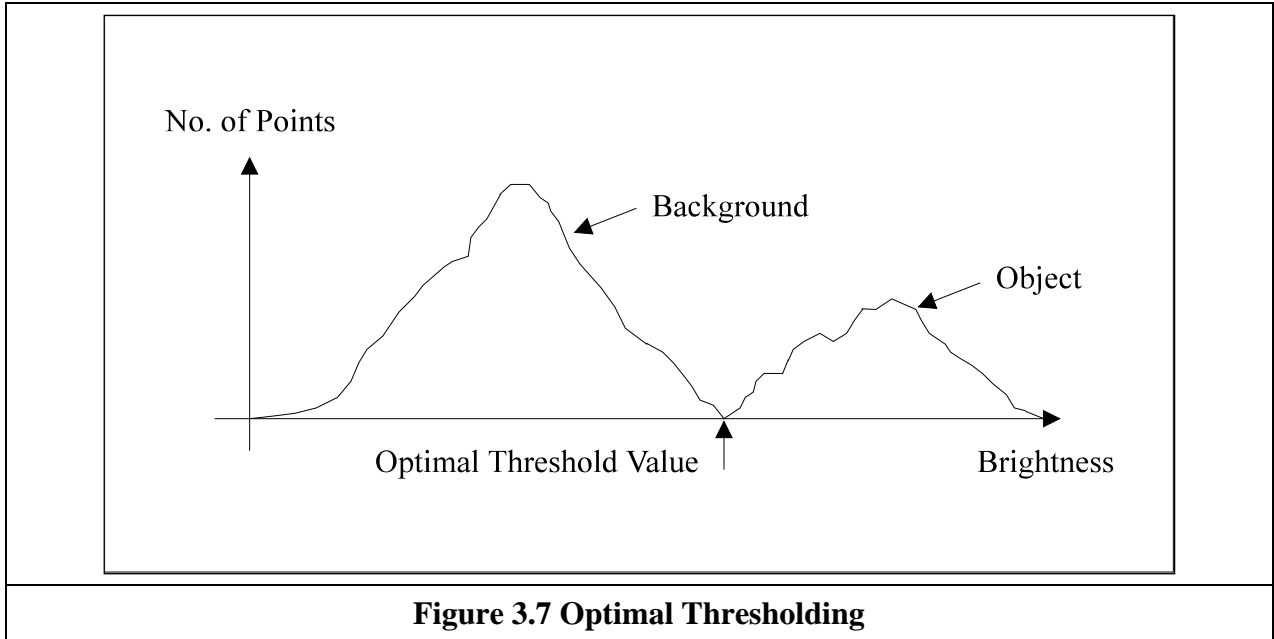
Given the original eye image, Figure 3.6 shows a thresholded image where all pixels above 160 brightness levels are set to white, and those below 160 brightness levels are set to black. By this process, the parts pertaining to the facial skin are separated from the background; the cheeks, forehead and other bright areas are separated from the hair and eyes. This can therefore provide a way of isolating points of interest.



**Figure 3.6 Thresholding the Eye Image**

Uniform thresholding clearly requires knowledge of the grey level, or the target features might not be selected in the thresholding process. If the level is not known, histogram equalisation or intensity normalisation can be used, but with the restrictions on performance stated earlier. This is, of course, a problem of image interpretation. These problems can only be solved by simple approaches, such as thresholding, for very special cases.





There are more advanced techniques, known as *optimal thresholding*. These usually seek to select a value for the threshold that separates an object from its background. This suggests that the object has a different range of intensities to the background, in order that an appropriate threshold can be chosen, as illustrated in Figure 3.7. *Otsu's method* [Otsu79] is one of the most popular techniques of optimal thresholding; there have been surveys [Sahoo88, Lee90, Glasbey93] which compare the performance different methods can achieve. Essentially, Otsu's technique maximises the likelihood that the threshold is chosen so as to split the image between an object and its background. This is achieved by selecting a threshold that gives the best separation of classes, for all pixels in an image. The basis is use of the normalised histogram where the number of points at each level is divided by the total number of points in the original image. As such, this represents a probability distribution for the intensity levels as

$$p(l) = \frac{\mathbf{O}(l)}{N^2} \quad (3.12)$$

This can be used to compute then zero- and first-order cumulative moments of the normalised histogram up to the  $k^{\text{th}}$  level as

$$\omega(k) = \sum_{l=1}^k p(l) \quad (3.13)$$

and

$$\mu(k) = \sum_{l=1}^k l \cdot p(l) \quad (3.14)$$

The total mean level  $\mu_T$  of the image is given by

$$\mu_T = \sum_{l=1}^{N_{\max}} l \cdot p(l) \quad (3.15)$$

The variance of the class separability (the similarity between the variables of the same class) is then the ratio

$$\sigma_B^2(k) = \frac{(\mu_T \cdot \omega(k) - \mu(k))^2}{\omega(k)(1 - \omega(k))} \quad \forall k \in 1, N_{\max} \quad (3.16)$$

The optimal threshold is the level for which the variance of class separability is at its maximum, namely the optimal threshold  $T_{\text{opt}}$  is that for which the variance

$$\sigma_B^2(T_{\text{opt}}) = \max_{1 \leq k < N_{\text{max}}} (\sigma_B^2(k)) \quad (3.17)$$

The implementation of the optimal thresholding is shown in Code 3.5. The code finds the optimum threshold in two stages. First, it uses the histogram of the input image to compute the moments in Equations 3.13 and 3.14. A second step computes the separability measure in Equation 3.16. The code keeps all the separability values for display, but in practice we only require to keep the maximum value.

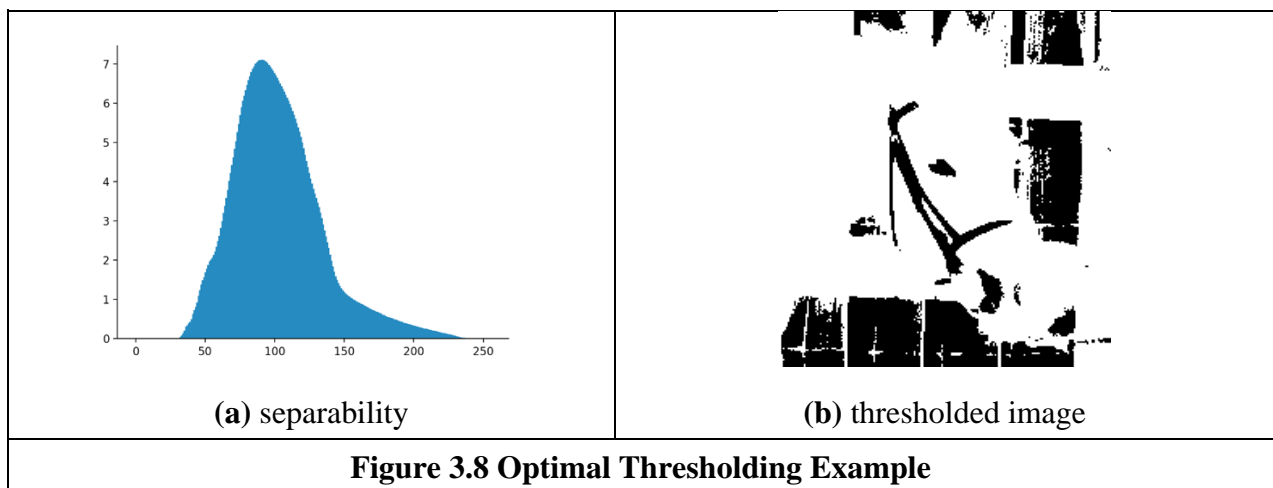
```
# Obtain histograms
normalization = 1.0 / float(width * height)
w[0] = normalization * inputHistogram[0]
for level in range(1, 256):
    w[level] = w[level-1] + normalization * inputHistogram[level]
    m[level] = m[level-1] + level * normalization * inputHistogram[level]

# Look for the maximum
maximumLevel = 0
for level in range(0, 256):
    if w[level] * (float(level) - w[level]) != 0:
        separability[level] = float(pow( ( m[255] * w[level] - m[level]), 2) \
            / (w[level] * (float(level) - w[level])))

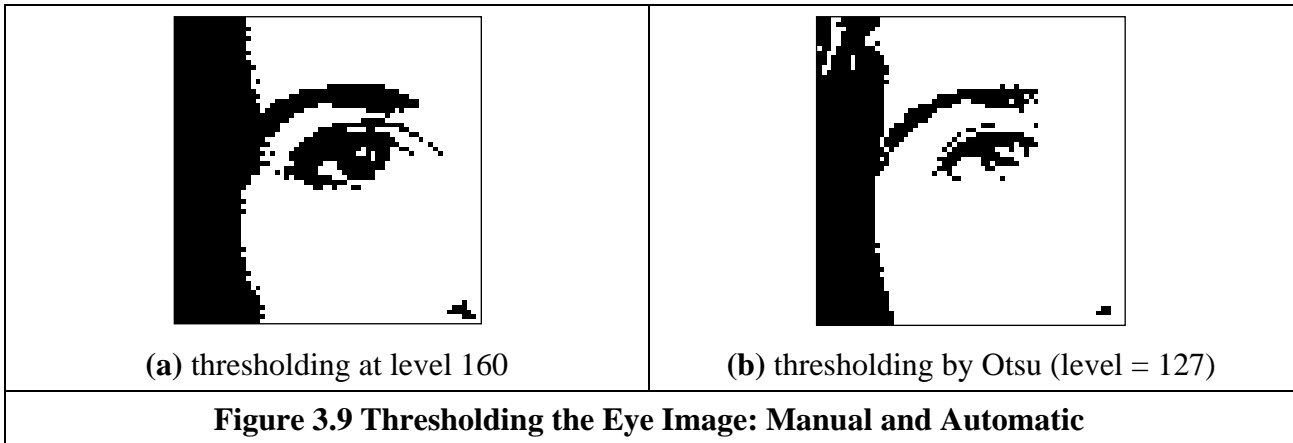
    if separability[level] > separability[maximumLevel]:
        maximumLevel = level
```

**Code 3.5 Optimal Thresholding**

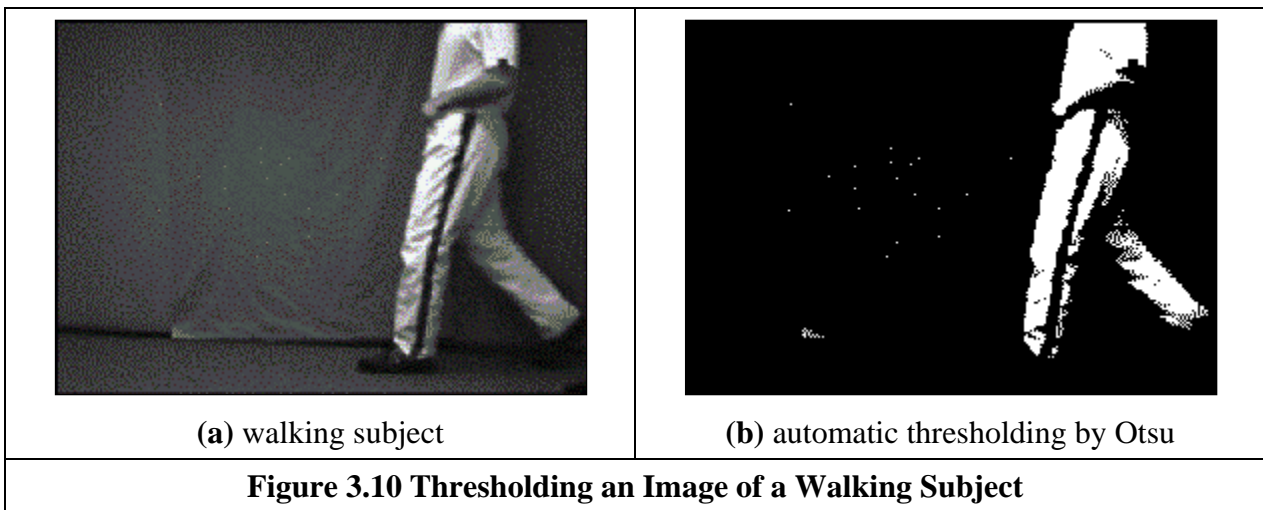
Figure 3.8 shows an example of optimal thresholding obtained with Code 3.5. Figure 3.8(a) shows the separability computed for each potential threshold. Low values represent thresholds that produce two regions with high inter class variance and high values minimize the interclass variance. The threshold for the maximum value is used to produce the thresholded image in Figure 3.8(b). In this figure the pixels are divided in two classes whose variance is minimum compared with any other possible threshold.



**Figure 3.8 Optimal Thresholding Example**



A comparison of uniform thresholding with optimal thresholding is given in Figure 3.9 for the eye image. The threshold selected by Otsu's operator is actually lower than the value selected manually, and so the thresholded image does omit some detail around the eye, especially in the eyelids. However, the selection by Otsu is automatic, as opposed to manual and this can be to application advantage in automated vision. Consider for example the need to isolate the human figure in Figure 3.10(a). This can be performed automatically by Otsu as shown in Figure 3.10(b). Note however that there are some extra points, due to illumination, which have appeared in the resulting image together with the human subject. It is easy to remove the isolated points, as we will see later, but more difficult to remove the connected ones. In this instance, the size of the human shape could be used as information to remove the extra points though you might like to suggest other factors that could lead to their removal.



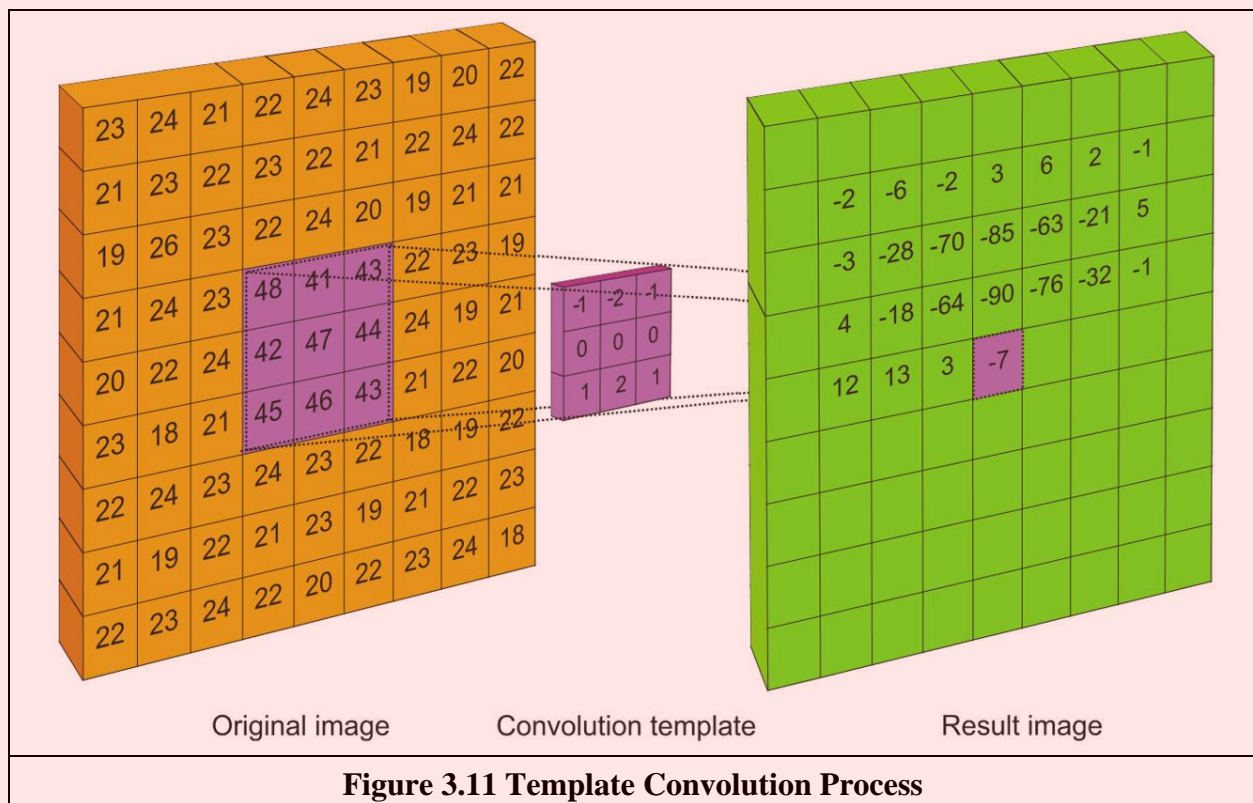
Also, we have so far considered global techniques, methods that operate on the entire image. There are also locally adaptive techniques that are often used to binarise document images prior to character recognition. As mentioned before, surveys of thresholding are available, and one approach [Rosin01] targets thresholding of images whose histogram is unimodal (has a single peak). One survey [Trier95] compares global and local techniques with reference to document image analysis. These techniques are often used in statistical pattern recognition: the thresholded object is classified according to its statistical properties. However, these techniques find less use in image interpretation, where a common paradigm is that there is more than one object in the scene, such as Figure 3.6 where the thresholding operator has selected many objects of potential interest. As such, only uniform thresholding is used in many vision applications, since objects are often occluded (hidden), and many objects have similar ranges of pixel intensity. Accordingly, more

sophisticated metrics are required to separate them, by using the uniformly thresholded image, as discussed in later Chapters. Further, the operation to process the thresholded image, say to fill in the holes in the silhouette or to remove the noise on its boundary or outside, is *morphology* which is covered later in Section 3.6. In general, it is often prudent to investigate the more sophisticated techniques of feature selection and extraction, to be covered later. Prior to that, we shall investigate group operators, which are a natural counterpart to point operators.

### 3.4 Group Operations

#### 3.4.1 Template Convolution

*Group operations* calculate new pixel values from a pixel's neighbourhood by using a 'grouping' process. The group operation is usually expressed in terms of *template convolution* where the template is a set of weighting coefficients. The template is usually square, and its size is usually odd to ensure that the result positioned precisely on a pixel. The size is usually used to describe the template; a  $3 \times 3$  template is three pixels wide by three pixels long. New pixel values are calculated by placing the template at the point of interest. Pixel values are multiplied by the corresponding weighting coefficient and added to an overall sum. The sum (usually) evaluates a new value for the centre pixel (where the template is centred) and this becomes the pixel in the output image. If the template's position has not yet reached the end of a line, the template is then moved horizontally by one pixel and the process repeats.



**Figure 3.11 Template Convolution Process**

This is illustrated in Figure 3.11 where a new image is calculated from an original one, by template convolution. The calculation obtained by template convolution at the centre pixel of the template in the original image becomes the point in the output result image. Since the template cannot extend beyond the original image, a new value cannot be computed for points at the border of the result image. When the template reaches the end of a line, it is repositioned to proceed from the start of

the next line. The process is shown part way through the raster scan, the next pixel to be calculated would be derived from the nine points to the right of the current position of the centre point of the template, and stored to the right of the point containing -7. For a  $3 \times 3$  neighbourhood, Figure 3.12, nine weighting coefficients  $w_i$  are applied to points in the original image to calculate a point in the new image. The position of the new point (at the centre) is shaded in the template.

$w_0$	$w_1$	$w_2$
$w_3$	$w_4$	$w_5$
$w_6$	$w_7$	$w_8$

**Figure 3.12  $3 \times 3$  Template and Weighting Coefficients**

To calculate the value in new image,  $\mathbf{N}$ , at point with co-ordinates  $x, y$ , the template in Figure 3.12 operates on an original image  $\mathbf{O}$  according to:

$$\mathbf{N}_{x,y} = \sum_{i \in \text{template}} \sum_{j \in \text{template}} w_{i,j} \times \mathbf{O}_{x(i),y(j)} \quad (3.18)$$

where the coordinates of the image point  $x(i), y(j)$  denote the position of the point that matches the weighting coefficient position. Note that we cannot ascribe values to the picture's *borders*. This is because when we place the template at the border, parts of the template fall outside the image and have no information from which to calculate the new pixel value. The width of the border equals half the size of the template. In Figure 3.11 the single pixel border points have been left blank. To calculate values for the border pixels, we have three choices:

1. set the border to black (or deliver a smaller picture);
2. assume (as in Fourier) that the image replicates to infinity along both dimensions and calculate new values by cyclic shift from the far border; or
3. calculate the border pixel value from a smaller area.

None of these approaches is optimal. The results in this book use the first option and set border pixels to black. Note that in many applications the object of interest is imaged centrally or, at least, imaged within the picture. As such, the border information is of little consequence to the remainder of the process. Here, the border points are set to black, by starting functions with a zero function which sets all the points in the picture initially to black (0).

An alternative representation for this process is given by using the convolution notation as

$$\mathbf{N} = \mathbf{W} * \mathbf{O} \quad (3.19)$$

where  $\mathbf{N}$  is the new image which results from convolving the template  $\mathbf{W}$  (of weighting coefficients) with the image  $\mathbf{O}$ .

The Matlab implementation of a template convolution operator `template_convolve` is given in Code 3.6. This function accepts, as arguments, the picture `image` and the template to be convolved with it, `template`. The result of template convolution is an image `convolved`. The operator first sets the resulting image to black (zero brightness levels). The widths `tc` and `tr` give the range of picture points to be processed in the outer `for` loops that give the co-ordinates of all points resulting from template convolution. The template is convolved at each picture point by generating a running summation of the pixel values within the template's window multiplied by the respective template weighting coefficient.

```
function convolved = template_convolve(image,template)
```

```

%get image dimensions
[rows,cols]=size(image);
%get template dimensions
[trows,tcols]=size(template);

%half of template rows is
tr=floor(trows/2);
%half of template cols is
tc=floor(tcols/2);

%set an output as black
convolved(1:rows,1:cols)=0;

%then convolve the template
for x = tc+1:cols-tc %address all columns except border
    for y = tr+1:rows-tr %address all rows except border
        sum=0; %initialise the sum
        for iwin=1:tcols %address all points in the template
            for jwin=1:trows
                sum=sum+image(y+jwin-tr-1,x+iwin-tc-1)*... % sum, Eq. 3.18
                    template(trows-jwin+1,tcols-iwin+1);
            end
        end
        convolved(y,x)=sum; %store as new point
    end
end
end

```

**Code 3.6 Template Convolution Operator**

Note that according to Eqn. 2.10, for convolution one of the signals is inverted along its principal axis. For images, convolution requires inversion along both axes which is why the template's arguments are inverted in Code 3.6. We shall consider convolution again in the next Section, via the frequency domain, and in Section 5.3.2.

Template convolution can of course be implemented in hardware and requires a two-line store, together with some further latches, for the (input) video data. The output is the result of template convolution, summing the result of multiplying weighting coefficients by pixel values. This is called pipelining, since the pixels are essentially move along a pipeline of information. Note that two line-stores can be used if the video fields only are processed. To process a full frame, one of the fields must be stored if it is presented in interlaced format. Processing can be analog, using operational amplifier circuits and Charge Coupled Device (CCD) for storage along bucket brigade delay lines. Finally, an alternative implementation is to use a parallel architecture: for Multiple Instruction Multiple Data (MIMD) architectures, the picture can be split into blocks (spatial partitioning); Single Instruction Multiple Data (SIMD) architectures can implement template convolution as a combination of shift and add instructions.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

**Figure 3.13 3×3 Averaging Operator Template Coefficients**

### 3.4.2 Averaging Operator

For an *averaging operator*, the template weighting functions are unity (or  $1/9$  to ensure that the result of averaging nine white pixels is white, not more than white!). The template for a  $3 \times 3$  averaging operator, implementing Equation 3.18, is given by the template in Figure 3.13 where the location of the point of interest is again shaded. The averaging operator is then

$$N_{x,y} = \frac{1}{MN} \sum_{i \in M} \sum_{j \in N} O_{x(i),y(j)} \quad (3.20)$$

where  $x(i), y(j)$  are the coordinates of image points within the template and  $M, N$  are the numbers of columns and rows in the template. The result of averaging an image with a  $9 \times 9$  operator is shown in Figure 3.14. This shows that much of the detail has now disappeared revealing the broad image structure. In order to implement averaging by using the template convolution operator, we need to define a template and then convolve it with the image (note also that there is an averaging operator `mean` in Matlab that can be used for this purpose).

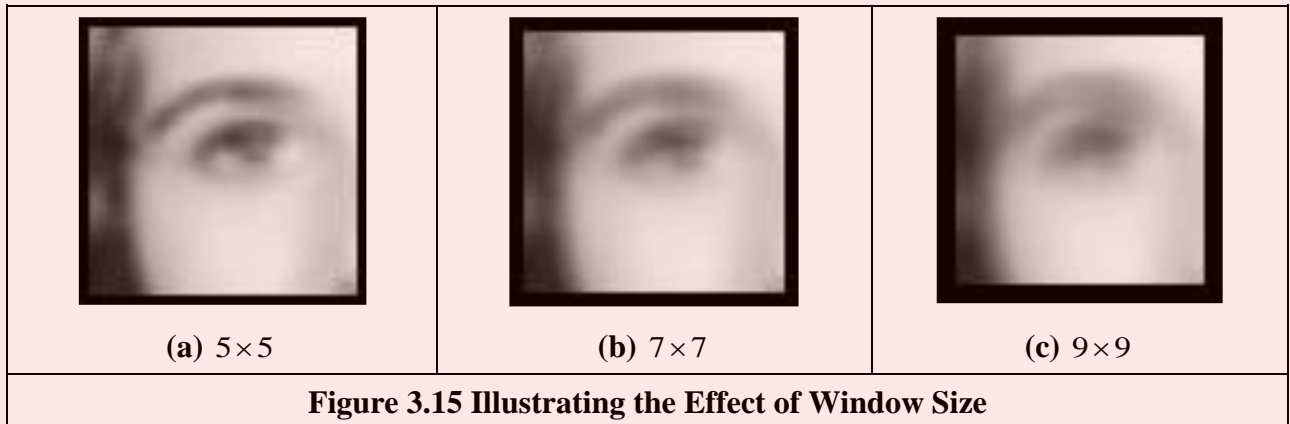


The effect of averaging is to reduce noise, this is its advantage. An associated disadvantage is that averaging causes blurring which reduces detail in an image. It is also a low pass filter since its effect is to allow low spatial frequencies to be retained, and to suppress high frequency components. A larger template, say  $9 \times 9$  or  $15 \times 15$ , will remove more noise (high frequencies) but reduce the level of detail. The size of an averaging operator is then equivalent to the reciprocal of the bandwidth of a low-pass filter it implements

### 3.4.3 On Different Template Size

Templates can be larger than  $3 \times 3$ . Since they are usually centred on a point of interest, to produce a new output value at that point, they are usually of odd dimension. For reasons of speed, the most common sizes are  $3 \times 3$ ,  $5 \times 5$  and  $7 \times 7$ . Beyond this, say  $9 \times 9$ , many template points are used to calculate a single value for a new point, and this imposes high computational cost, especially for large images. (For example, a  $9 \times 9$  operator covers 9 times more points than a  $3 \times 3$  operator.) Square templates have the same properties along both image axes. Some implementations use vector templates (a line), either because their properties are desirable in a particular application, or for reasons of speed.

The effect of larger averaging operators is to smooth the image more, to remove more detail whilst giving greater emphasis to the large structures. This is illustrated in Figure 3.15. A  $5 \times 5$  operator, Figure 3.15(a), retains more detail than a  $7 \times 7$  operator, Figure 3.15(b), and much more than a  $9 \times 9$  operator, Figure 3.15(c). Conversely, the  $9 \times 9$  operator retains only the largest structures such as the eye region (and virtually removing the iris) whereas this is retained more by the operators of smaller size. Note that the larger operators leave a larger border (since new values cannot be computed in that region) and this can be seen in the increase in border size for the larger operators, in Figures 3.15(b) and (c).



### 3.4.4 Template Convolution via the Fourier Transform

The Fourier transform actually gives an alternative method to implement template convolution and to speed it up, for larger templates. The question to be answered here is ‘how big?’. In Fourier transforms, the process that is dual to *convolution* is multiplication (as in Section 2.3). So template convolution (denoted  $*$ ) can be implemented by multiplying the Fourier transform of the template  $\mathfrak{F}(\mathbf{T})$  with the Fourier transform of the picture,  $\mathfrak{F}(\mathbf{P})$ , to which the template is to be applied. It is perhaps a bit confusing that we appear to be multiplying matrices, but the multiplication is point-by-point in that the result at each point is that of multiplying the (single) points at the same positions in the two matrices. The result needs to be inverse transformed to return to the picture domain.

$$\mathbf{P} * \mathbf{T} = \mathfrak{F}^{-1}(\mathfrak{F}(\mathbf{P}) \cdot \mathfrak{F}(\mathbf{T})) \quad (3.21)$$

The transform of the template and the picture need to be the same size before we can perform the point by point multiplication ( $\cdot$ ). Accordingly, the image containing the template is zero-padded prior to its transform which simply means that zeroes are added to the template which lead to a template of the same size as the image. The process is illustrated in Code 3.7(a) and starts by calculation of the transforms of the image and of the zero-padded template. Then, the transform of the template is multiplied by the transform of the picture point-by-point (using the  $\cdot *$  operator). (Theoretical study of this process is presented in Section 5.3.2 where we show how the same process can be used to find shapes in images.) Finally, the inverse Fourier transform is used to deliver the result. Code 3.7(b) shows an implementation in Python. This code computes the summation defining the Fourier transform by performing an iteration. First, the template is flipped and padded. Afterwards, the coefficients are obtained by performing the multiplication of the complex numbers of the image and of the template coefficients.

```
image_eye=imread('eye_orig.jpg');
```



```

image_eye=double(image_eye(:,:,1));
image_transform=fft2(image_eye);
template_transform=fft2(pad(image_eye, ave_template(7)));
inverted_transform=ifft2(rearrange(image_transform.*template_transform));

```

The transform based implementation of direct averaging can be combined as

```

averaged_image=ifft2(rearrange(fft2(eye).*fft2(pad(eye,ave_template(7)))));

```

(a) Matlab

```

# Padding
widthPad, heightPad = width+kernelSize-1, height+kernelSize-1

templatePadFlip = createImageF(widthPad, heightPad)
for x,y in itertools.product(range(0, kernelSize), range(0, kernelSize)):
    templatePadFlip[y, x] = kernelImage[kernelSize-y-1, kernelSize-x-1]

# Compute coefficients
imageCoeff, maxFrequencyW, maxFrequencyH = computeCoefficients(inputPad)
templateCoeff, _, _ = computeCoefficients(templatePadFlip)

# Frequency domain multiplication
for kw,kh in itertools.product(range(-maxFrequencyW, maxFrequencyW + 1), \
                               range(-maxFrequencyH, maxFrequencyH + 1)):
    w = kw + maxFrequencyW
    h = kh + maxFrequencyH

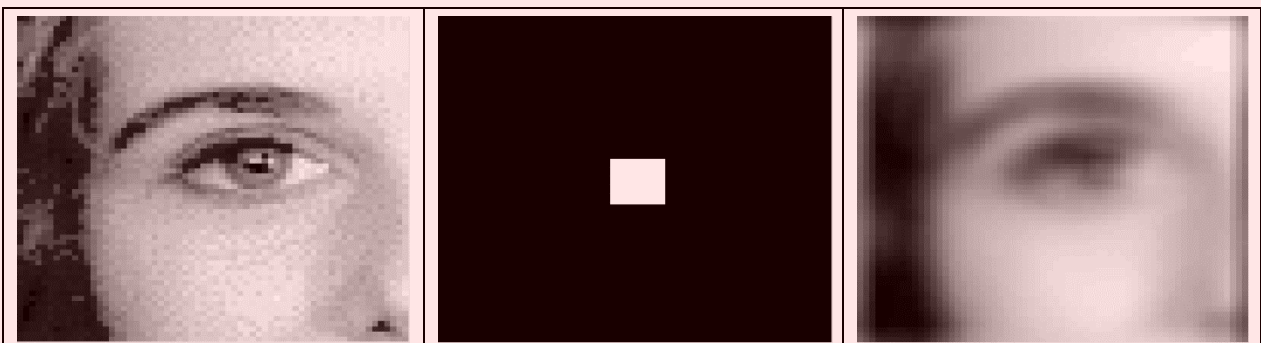
    resultCoeff[h,w][0] = (imageCoeff[h,w][0] * templateCoeff[h,w][0] - \
                          imageCoeff[h,w][1] * templateCoeff[h,w][1])
    resultCoeff[h,w][1] = (imageCoeff[h,w][1] * templateCoeff[h,w][0] + \
                          imageCoeff[h,w][0] * templateCoeff[h,w][1])

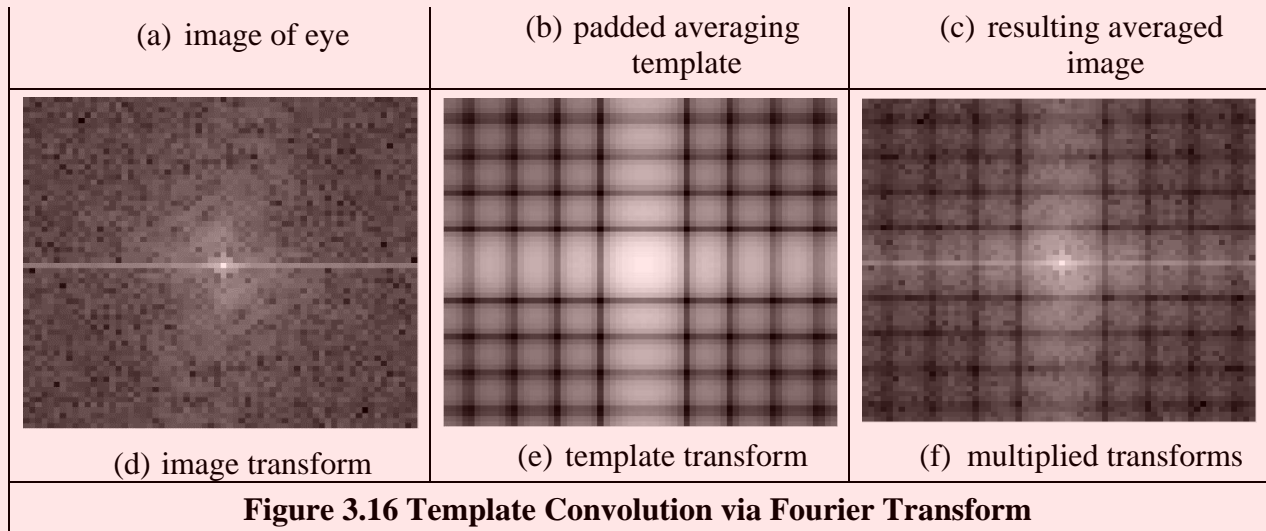
```

(b) Python

### Code 3.7 Template Convolution via the Fourier Transform

Code 3.7 is simply a different implementation of direct averaging. It achieves a similar result, but by transform domain calculus. The operation is shown in Figure 3.16: an image of the eye (a) is transformed to give (d); the averaging template is padded to the same size as the image (b) and transformed (e); the multiplied transforms (f) are inverse transformed to give an averaged version of the eye (c). There is one major difference between the Fourier and the direct implementations: the borders of the images differ (where the *border* is of width equal to one half of the template's width). This is because for direct averaging the border points are set to zero whereas in the Fourier implementation the image is assumed to replicate to infinity, as in Equation 2.26. (The *rearrange* function, Equation 2.30, is used since the padding function places the template at the centre of the image). Note that the template transform is a 2D sinc function viewed as an image and that the *logarithm* of the magnitude (Section 3.3.1) has been used to display all transforms.





It can be faster to use the transform-based implementation (Code 3.7) rather than the direct implementation (Code 3.6), depending on the size of the template. For a square template with  $N \times N$  points the computational cost of a 2D FFT is of the order of  $2N^2 \log_2(N)$ . If the transform of the template is pre-computed, there are two transforms required and there is one multiplication for each of the  $N^2$  transformed points. The total cost of the Fourier implementation of template convolution is then of the order of

$$C_{FFT} = 4N^2 \log_2(N) + N^2 \quad (3.22)$$

The cost of the direct implementation for a  $m \times m$  template is then  $m^2$  multiplications for each image point, so the cost of the direct implementation is of the order of

$$C_{dir} = N^2 m^2 \quad (3.23)$$

For  $C_{dir} < C_{FFT}$ , we require:

$$N^2 m^2 < 4N^2 \log_2(N) + N^2 \quad (3.24)$$

If the direct implementation of template matching is faster than its Fourier implementation, we need to choose  $m$  so that

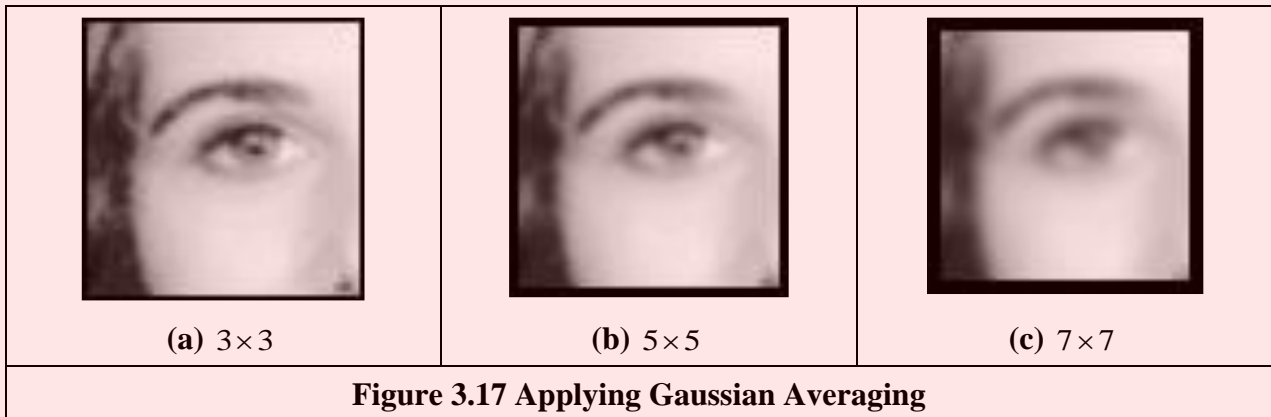
$$m^2 < 4 \log_2(N) + 1 \quad (3.25)$$

This implies that, for a  $256 \times 256$  image, a direct implementation is fastest for  $3 \times 3$  and  $5 \times 5$  templates, whereas a transform-based calculation is faster for larger ones. An alternative analysis [Campbell69] suggested that [Gonzalez17] ‘if the number of non-zero terms in (the template) is less than 132 then a direct implementation .... is more efficient than using the FFT approach’. In OpenCV, for some versions the limit appears to  $7 \times 7$  [OpenCV-TM] for using the FFT and for a kernel size of  $5 \times 5$  or less a direct version is used. An  $11 \times 11$  operator is a considerably larger template than our analysis suggests, whereas OpenCV has the same limit as the analysis here. This might be due to higher considerations of complexity than our analysis has included. There are, naturally, further considerations in the use of transform calculus, the most important being the use of windowing (such as Hamming or Hanning) operators to reduce variance in high order spectral estimates. This implies that template convolution by transform calculus should be used when large templates are involved, and when speed is critical. If speed is indeed critical, it might be prudent to implement the operator in dedicated hardware, as described earlier.

### 3.4.5 Gaussian Averaging Operator

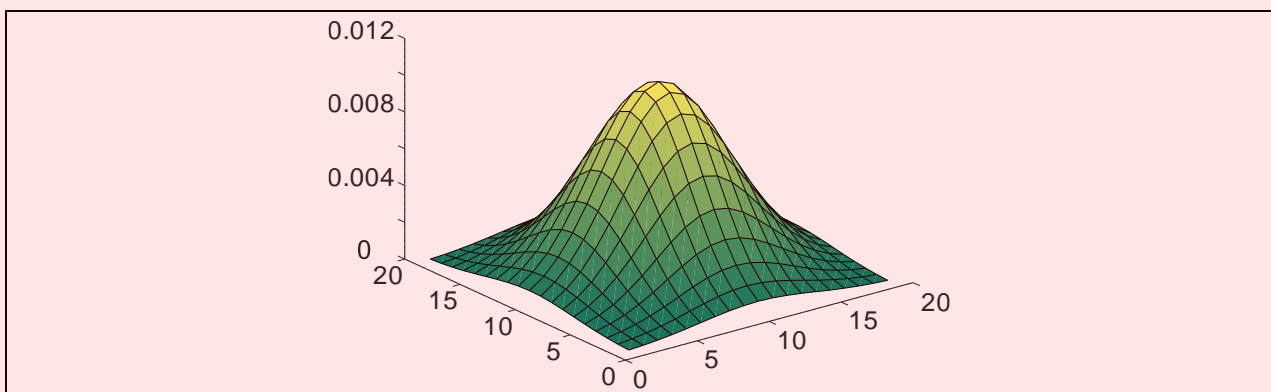
The *Gaussian averaging operator* has been considered to be optimal for image smoothing. The template for the Gaussian operator has values set by the Gaussian relationship. The *Gaussian function*  $g$  at coordinates  $x,y$  is controlled by the *variance*  $\sigma^2$  according to:

$$g(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{3.26}$$



**Figure 3.17 Applying Gaussian Averaging**

Equation 3.26 gives a way to calculate coefficients for a Gaussian template which is then convolved with an image. The effects of selection of Gaussian templates of differing size are shown in Figure 3.17. The Gaussian function essentially removes the influence of points greater than  $3\sigma$  in (radial) distance from the centre of the template. The  $3 \times 3$  operator, Figure 3.17(a), retains many more of the features than those retained by direct averaging (Figure 3.15). The effect of larger size is to remove more detail (and noise) at the expense of losing features. This is reflected in the loss of internal eye component by the  $5 \times 5$  and the  $7 \times 7$  operators in Figures 3.17(b) and (c), respectively.



**Figure 3.18 Gaussian Function**

A surface plot of the 2D Gaussian function of Equation 3.26 has the famous bell-shape, as shown in Figure 3.18 (for a window size of  $19 \times 19$  and standard deviation of 4.0). The values of the function at discrete points are the values of a Gaussian template. Convolving this template with an image gives Gaussian averaging: the point in the averaged picture is calculated from the sum of a region where the central parts of the picture are weighted to contribute more than the peripheral points. The size of the template essentially dictates appropriate choice of the variance. The variance is

chosen to ensure that template coefficients drop to near zero at the template's edge. The template for size  $5 \times 5$  with variance unity is shown in Figure 3.19.

0.002	0.013	0.022	0.013	0.002
0.013	0.060	0.098	0.060	0.013
0.022	0.098	0.162	0.098	0.022
0.013	0.060	0.098	0.060	0.013
0.002	0.013	0.022	0.013	0.002

**Figure 3.19 Template for the  $5 \times 5$  Gaussian Averaging Operator ( $\sigma = 1.0$ )**

When a Gaussian template is convolved with an image, it produces Gaussian blurring. It is actually possible to give the Gaussian blurring function antisymmetric properties by scaling the  $x$  and  $y$  coordinates. This can find application when an object's shape, and orientation, is known prior to image analysis.

By reference to Figure 3.17 it is clear that the Gaussian filter can offer improved performance compared with direct averaging: more features are retained whilst the noise is removed. This can be understood by Fourier transform theory. In Section 2.5.2 we found that the Fourier transform of a square is a two-dimensional sinc function. This has a frequency response where the magnitude of the transform does not reduce in a smooth manner and has regions where it becomes negative, called sidelobes. These can have undesirable effects since there are high frequencies that contribute more than some lower ones, a bit paradoxical in low-pass filtering which aims to remove noise. In contrast, the Fourier transform of a Gaussian function is another Gaussian function, which decreases smoothly without these sidelobes. This can lead to better performance since the contributions of the frequency components reduce in a controlled manner.

```

centre = (kernelSize - 1) / 2
sumValues = 0
for x,y in itertools.product(range(0, kernelSize), range(0, kernelSize)):
    kernelImage[y,x] = math.exp( -0.5 * (math.pow((x - centre)/sigma, 2.0) + \
                                math.pow((y - centre)/sigma, 2.0)) )
    sumValues += kernelImage[y,x]

# Normalisation
for x,y in itertools.product(range(0, kernelSize), range(0, kernelSize)):
    kernelImage[y,x] /= sumValues

```

**Code 3.8 Gaussian Template Specification**

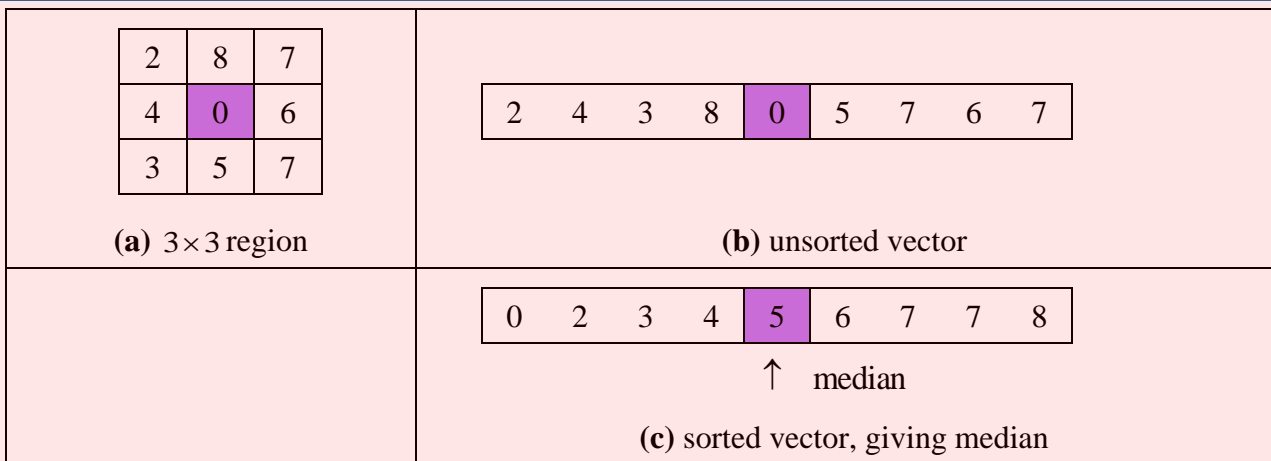
In a software implementation of the Gaussian operator, we need a function implementing Equation 3.26, the `Gaussian_template` function in Code 3.8. This is used to calculate the coefficients of a template to be centred on an image point. The two arguments are `winSize`, the (square) operator's size, and the standard deviation `sigma` that controls its width, as discussed earlier. The operator coefficients are normalised by the sum of template values, as before. This summation is stored in `sum`, which is initialised to zero. The centre of the square template is then evaluated as half the size of the operator. Then, all template coefficients are calculated by a version of Equation 3.26 which specifies a weight relative to the centre co-ordinates. Finally, the normalised template coefficients are returned as the Gaussian template. The operator is used in template convolution, via `convolve`, as in direct averaging (Code 3.6) or by Fourier (Code 3.7).

### 3.4.6 More on Averaging

There is more than could be discussed on basic smoothing, e.g. smoothing was earlier achieved by low-pass filtering via the Fourier transform (Section 2.8), but we shall move on to other operators. The averaging process is actually a statistical operator since it aims to estimate the mean of a local neighbourhood. The *error* in the process is naturally high, for a population of  $N$  samples, the statistical error is of the order of:

$$error = \frac{mean}{\sqrt{N}} \tag{3.27}$$

Increasing the averaging operator’s size improves the error in the estimate of the mean, but at the expense of fine detail in the image. The average is of course an estimate optimal for a signal corrupted by *additive Gaussian noise*. The estimate of the mean maximised the probability that the noise has its mean value, namely zero. According to the *central limit theorem*, the result of adding many noise sources together is a Gaussian distributed noise source. In images, noise arises in sampling, in quantisation, in transmission and in processing. By the central limit theorem, the result of these (independent) noise sources is that image noise can be assumed to be Gaussian. In fact, image noise is not necessarily Gaussian-distributed, giving rise to more statistical operators. One of these is the median operator which has demonstrated capability to reduce noise whilst retaining feature boundaries (in contrast to smoothing which blurs both noise and the boundaries), and the mode operator which can be viewed as optimal for a number of noise sources, including *Rayleigh noise*, but is very difficult to determine for small, discrete, populations.



**Figure 3.20 Finding the Median from a 3×3 Template.**

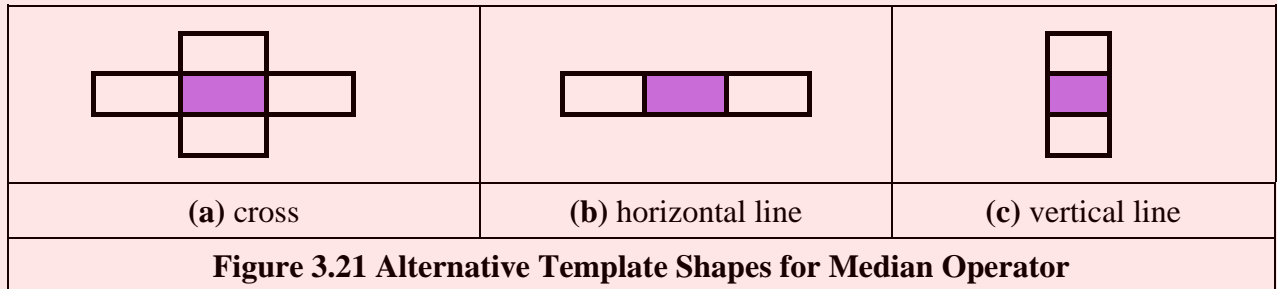
## 3.5 Other Image Processing Operators

### 3.5.1 Median Filter

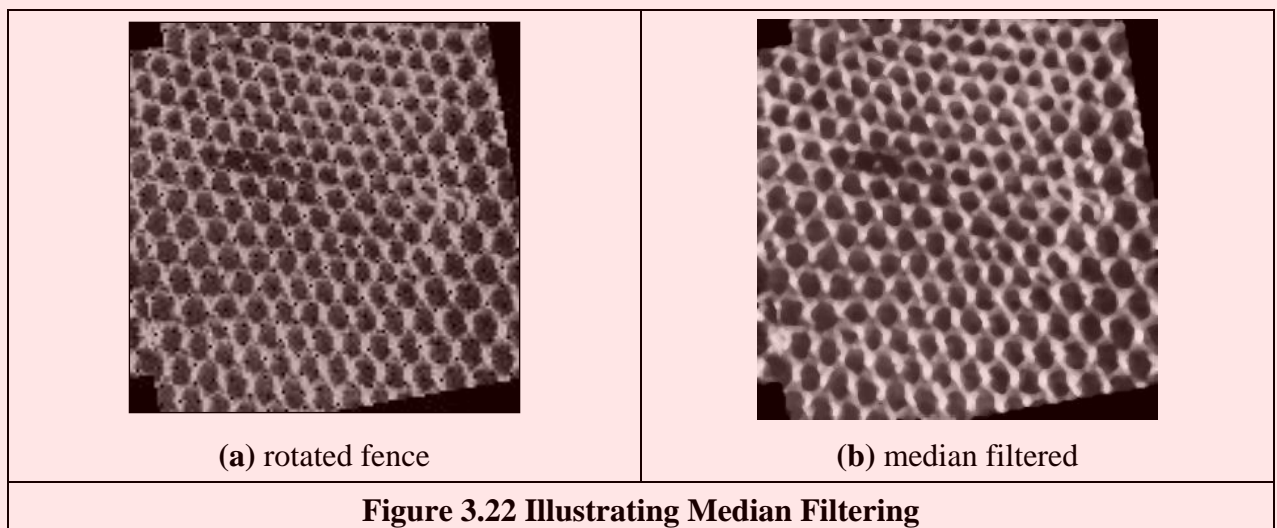
The *median* is another frequently used statistic; the median is the centre of a rank-ordered distribution. The median is usually taken from a template centred on the point of interest. Given the arrangement of pixels in Figure 3.20(a), the pixel values are arranged into a vector format, Figure 3.20(b). The vector is then sorted into ascending order, Figure 3.20(c). The median is the central component of the sorted vector, this is the fifth component since we have nine values.

The median can of course be taken from larger template sizes. The development here has aimed not only to demonstrate how the median operator works, but also to provide a basis for further

development. The rank ordering process is computationally demanding (slow) and motivates study into the deployment of fast algorithms, such as Quicksort, (e.g. [Huang79] is an early approach), though other approaches abound [Weiss06]. The computational demand also has motivated use of template shapes, other than a square. A selection of alternative shapes is shown in Figure 3.21. Common alternative shapes include a cross or a line (horizontal or vertical), centred on the point of interest, which can afford much faster operation since they cover fewer pixels. The basis of the arrangement presented here could be used for these alternative shapes, if required.



The median has a well-known ability to remove *salt and pepper noise*. This form of noise, arising from say decoding errors in picture transmission systems, can cause isolated white and black points to appear within an image. It can also arise when rotating an image, when points remain unspecified by a standard rotation operator (Chapter 10), as in a texture image, rotated by  $10^\circ$  in Figure 3.22(a). When a median operator is applied, the salt and pepper noise points will appear at either end of the rank ordered list and are removed by the median process, as shown in Figure 3.22(b). The median operator has practical advantage, due to its ability to retain edges (the boundaries of shapes in images) whilst suppressing the noise contamination. As such, like direct averaging, it remains a worthwhile member of the stock of standard image processing tools. For further details concerning properties and implementation, have a peep at [Hodgson85]. (Note that practical implementation of image rotation is a Computer Graphics issue, and is usually by texture mapping; further details can be found in [Hearn97].)



Code 3.9 illustrates the process of applying a median filter to an image. Each pixel defines a list containing the values in a window region. The value of the pixel in the output image is obtained by finding the central element of the sorted list. The image in Figure 3.22 (b) was obtained using Code 3.9 with a kernel size parameter of 5.

```

for x,y in itertools.product(range(0, width), range(0, height)):
    region = [ ]
    for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):

        posY = y + wy - kernelCentre
        posX = x + wx - kernelCentre

        if posY > -1 and posY < height and posX > -1 and posX < width:
            region.append(inputImage[posY,posX])

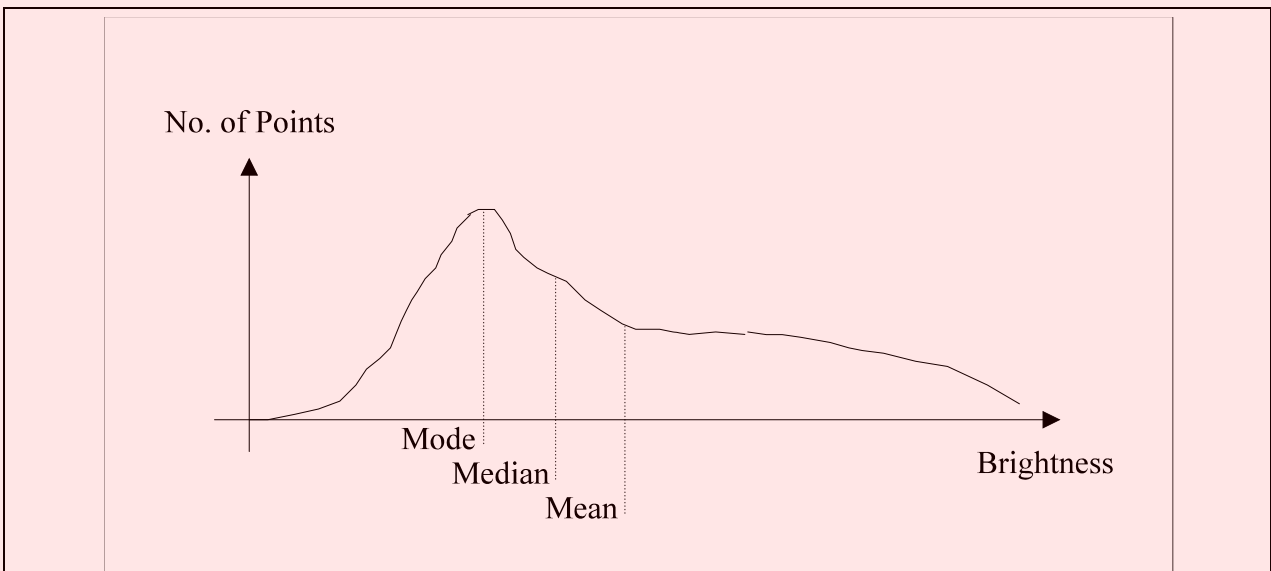
    numPixels = len(region)
    if numPixels > 0:
        region.sort()
        outputImage[y,x] = region[numPixels/2]
    
```

**Code 3.9 Median Filtering**

### 3.5.2 Mode Filter

The *mode* is the final statistic of interest, though there are more advanced filtering operators to come. The mode is of course very difficult to determine for small populations and theoretically does not even exist for a continuous distribution. Consider for example determining the mode of the pixels within a square 5×5 template. Naturally, it is possible for all 25 pixels to be different, so each could be considered to be the mode. As such we are forced to estimate the mode: the truncated median filter, as introduced by Davies [Davies88], aims to achieve this. The *truncated median filter* is based on the premise that for many non-Gaussian distributions, the order of the mean, the median and the mode is the same for many images, as illustrated in Figure 3.23.

Accordingly, if we truncate the distribution (i.e. remove part of it, where the part selected to be removed in Figure 3.23 is from the region beyond the mean) then the median of the truncated distribution will approach the mode of the original distribution.



**Figure 3.23 Arrangement of Mode, Median and Mean**

In implementation the operator first finds the mean and the median of the current window. The distribution of intensity of points within the current window is truncated on the side of the mean so that the median now bisects the distribution of the remaining points (as such not affecting

symmetrical distributions). So that the median bisects the remaining distribution, if the median is less than the mean, the point at which the distribution is truncated, *upper*, is

$$\begin{aligned} upper &= median + (median - \min(distribution)) \\ &= 2 \cdot median - \min(distribution) \end{aligned} \quad (3.28)$$

If the median is greater than the mean, then we need to truncate at a lower point (before the mean), *lower*, given by

$$lower = 2 \cdot median - \max(distribution) \quad (3.29)$$

The median of the remaining distribution then approaches the mode. The truncation is performed by storing pixels values in a vector. The median of the truncated vector is then the output of the truncated median filter at that point. Naturally, the window is placed at each possible image point, as in template convolution. However, there can be several iterations at each position to ensure that the mode is approached. In practice only few iterations are usually required for the median to converge to the mode. The window size is usually large, say  $7 \times 7$  or  $9 \times 9$  or even more.

Code 3.10 illustrates the implementation of the operator. The current window pixels are contained in the `region` list. The `mean` and `median` variables store statistics of the region and they are used to compute the values of the `upper` and `lower` variables according to Equation 3.28 and Equation 3.29. These values are used to create the list `truncatedRegion` that contains the truncated vector. Finally, the centre point of this vector is used produce the output pixel value.

The action of the operator is illustrated in Figure 3.24 when applied to a  $128 \times 128$  part of the ultrasound image (Figure 1.1(c)), from the centre of the image and containing a cross-sectional view of an artery. Ultrasound results in particularly noisy images, in part because the scanner is usually external to the body. The noise is actually multiplicative Rayleigh noise for which the mode is the optimal estimate. This noise obscures the artery which appears in cross-section in Figure 3.24(a); the artery is basically elliptical in shape. The action of the  $9 \times 9$  truncated median operator, Figure 3.24(b) is to remove noise whilst retaining feature boundaries whilst a larger operator shows better effect, Figure 3.24(c). An extra example is shown in Figure 3.24 (d). This image contains added salt and pepper noise to highlight the usefulness of the filter. We can observe that a small kernel is enough to remove such noise. However, edge definition is lost. This problem is more evident for larger kernels.

```
kernelCentre = (kernelSize - 1) / 2
for x,y in itertools.product(range(0, width), range(0, height)):

    # Iterate Window to collect values to compute mean and median
    region = [ ]
    sumValues = 0
    for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):
        posY, posX = y + wy - kernelCentre, x + wx - kernelCentre

        if posY > -1 and posY < height and posX > -1 and posX < width:
            sumValues += inputImage[posY,posX]
            region.append(inputImage[posY,posX])

    # Compute mean and median of the window
    numPixels = len(region)
    if numPixels > 0:

        # Mean and median
        mean = sumValues / numPixels
        region.sort()
        median = region[numPixels/2]

        # Upper and low
        upper, lower = 2.0*median-region[0], 2.0*median-region[numPixels-1]
```



```

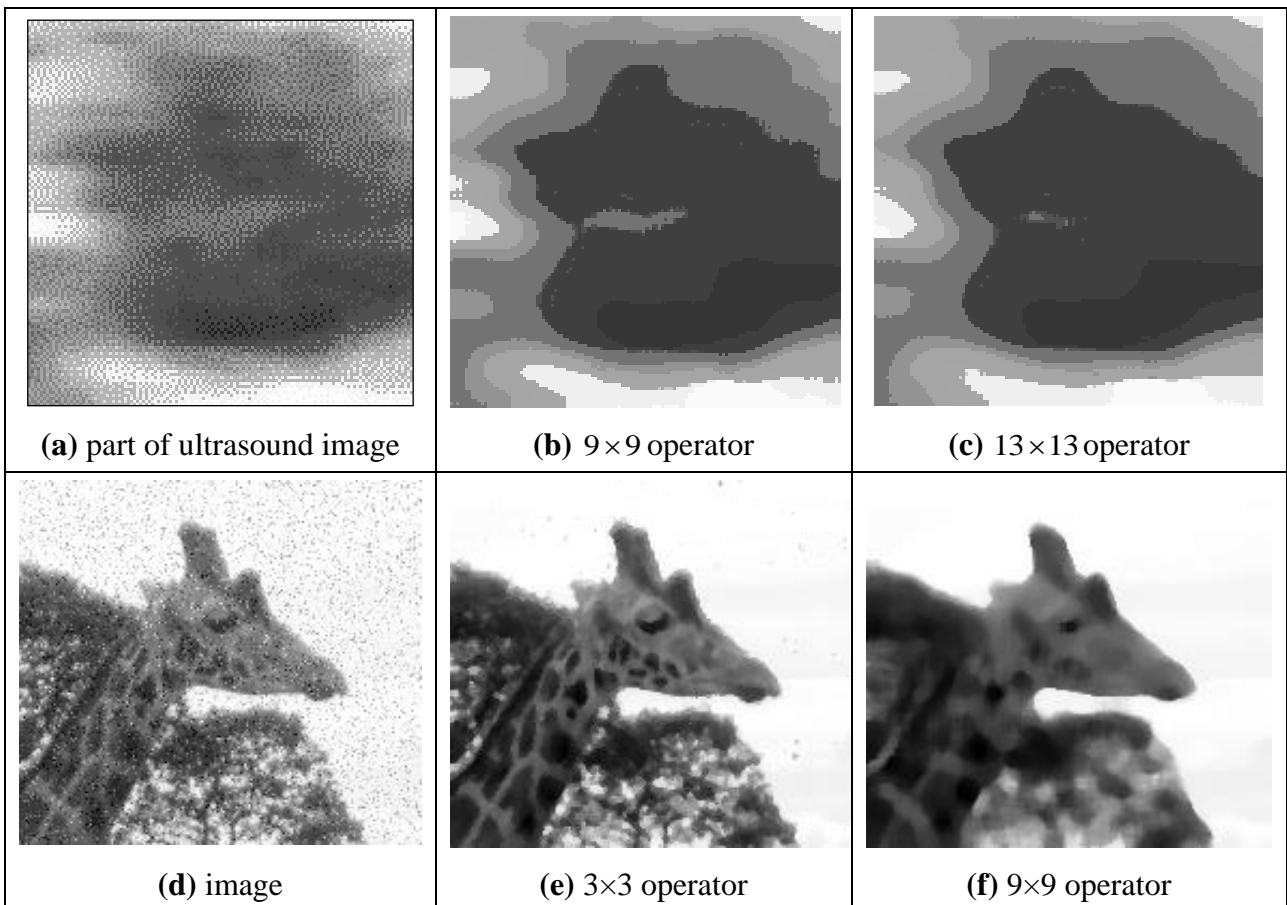
# Create a list of truncated values
truncatedRegion = [ ]
for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):
    posY, posX = y + wy - kernelCentre, x + wx - kernelCentre

    if posY > -1 and posY < height and posX > -1 and posX < width:
        if (inputImage[poSY,poSX] < upper and median < mean) or
            (inputImage[poSY,poSX] > lower and median > mean):
            truncatedRegion.append(inputImage[poSY,poSX])

# Compute median of truncated pixels
numTruncatedPixels = len(truncatedRegion)
if numTruncatedPixels > 0:
    truncatedRegion.sort()
    outputImage[y,x] = truncatedRegion[numTruncatedPixels/2]
else:
    outputImage[y,x] = median
    
```

**Code 3.10 Truncated Median Filtering**

Close examination of the result of the truncated median filter is that a selection of boundaries are preserved which are not readily apparent in the original ultrasound image. This is one of the known properties of median filtering: an ability to reduce noise whilst retaining feature boundaries. Indeed, there have actually been many other approaches to speckle filtering; the most popular for ultrasound include direct averaging [Shankar86], median filtering, adaptive (weighted) median filtering [Loupas87] and with basis using nonlocal means [Coupé09] and anisotropic (coherent) diffusion [Abd-Elmoniem 2002] (the latter two to be covered next).



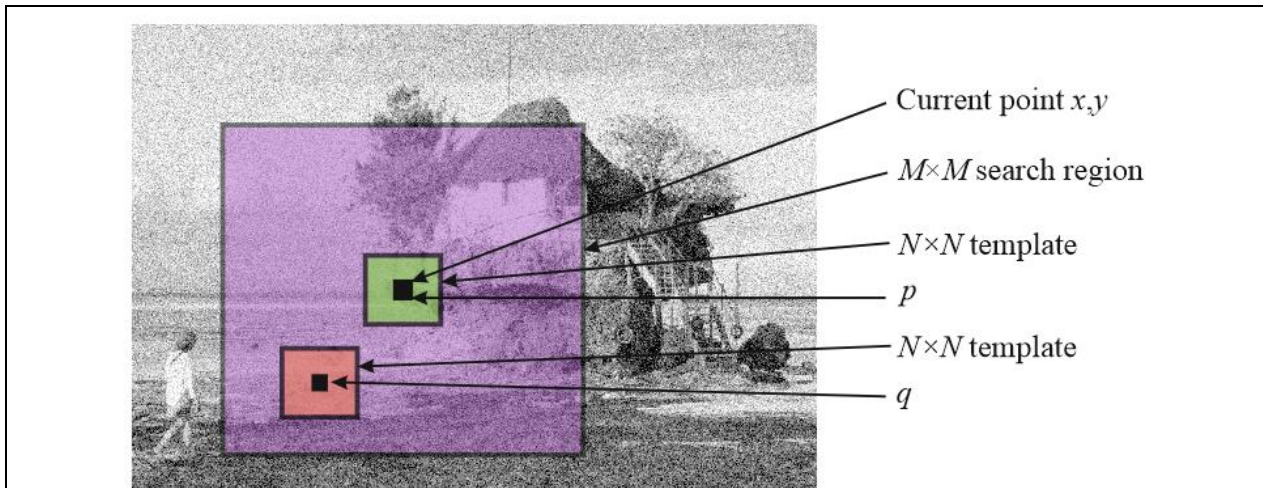
**Figure 3.24 Applying Truncated Median Filtering**

### 3.5.3 Non-Local Means

The *non-local means* operator [Buades05] is an extended version of the averaging operator. The basic function of the operator is to assign a point a value that is the mean of an area that is closest to the mean at the value of the point, rather than the mean at that point. As the original paper put it “the denoised value at  $x$  is a mean of the values of all points whose Gaussian neighbourhood looks like the neighbourhood of  $x$ ”. As the former this was rather hard to express and the latter is rather terse, it will be difficult to understand. So let’s slow down a bit. By denoting the average (mean) at point  $p$  as  $\bar{x}(p)$ , for an  $N \times N$  region, from Equation 3.20

$$\bar{x}(p) = \frac{1}{N^2} \sum_{i \in N} \sum_{j \in N} \mathbf{O}_{x(i),y(j)} \quad (3.30)$$

where  $x(i), y(j)$  are the coordinates of image points within the template. If this operation is performed over the whole image, we have applied the direct averaging operator. The parts of the process are shown in Figure 3.25.



**Figure 3.25 Non-Local Means Process**

As averaging processes use a weighted sum if we use the Gaussian function, via Equation 3.26, to calculate a weighting  $w$  between the average at point  $p$  and the average at point  $q$

$$w(p, q) = g(\bar{x}(p) - \bar{x}(q)) = \frac{1}{2\pi\sigma^2} e^{-\frac{(\bar{x}(p) - \bar{x}(q))^2}{2\sigma^2}} \quad (3.31)$$

Then the weight will be maximum when  $\bar{x}(p) = \bar{x}(q)$  and much less when the two averages are very different. The product  $\bar{x}(p) \times w(p, q)$  will be close to  $\bar{x}(p)$  when the mean of the point of interest  $p$  is the same as the mean of the region at point  $q$ . We shall use an  $M \times M$  search region. The non-local means operator is then a weighted summation

$$\begin{aligned} \mathbf{N}_{x,y} &= \frac{1}{k} \sum_{i \in M} \sum_{j \in M} \mathbf{O}_{x(i),y(j)} g(\bar{x}(p) - \bar{x}(q)) \\ &= \frac{1}{\sum_{i \in M} \sum_{j \in M} g(\bar{x}(p) - \bar{x}(q))} \sum_{i \in M} \sum_{j \in M} \mathbf{O}_{x(i),y(j)} g(\bar{x}(p) - \bar{x}(q)) \end{aligned} \quad (3.32)$$

where  $k$  is the normalising function  $k = \sum_{i \in M} \sum_{j \in M} g(\bar{x}(p) - \bar{x}(q))$ . The parameters that must be chosen are the window size of the averaging operator,  $N$ , the size of the search region,  $M$ , and the standard deviation. A Matlab implementation is given in Code 3.11 which follows the equations above. The border of the resulting image is set to half the window size of the larger of the averaging and range operators.

```
function filtered = non_local_means(image,winsize,searchsize,st_dev)

%get image dimensions
[rows,cols]=size(image);

%set the output image to black
filtered(1:rows,1:cols)=0;
local_mean(1:rows,1:cols)=0;

%half of template is
halfwin=floor(winsize/2);
%and half of the search
halfsearch=floor(searchsize/2);

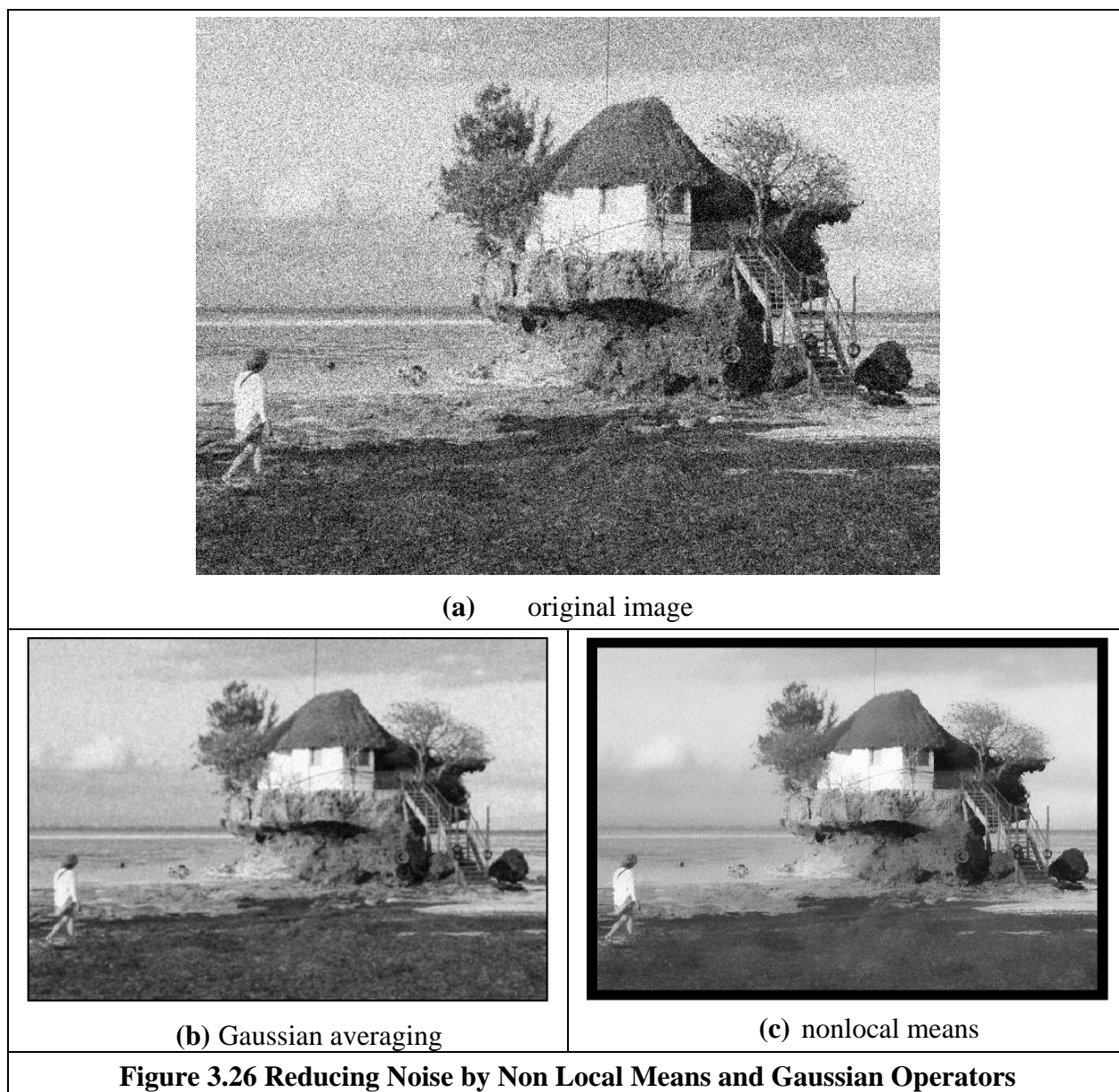
%process points according to largest window function
if winsize>searchsize
    border=halfwin; %normal case
else
    border=halfsearch; %which is possible, but daft
end

%%then form the local averages
local_mean=floor(ave(image,winsize));

%we start by looking at all image points except those in the border
for x = border+1:cols-border
    for y = border+1:rows-border
        %need to total up the weights (for later normalisation)
        weightsum=0;
        %need to total up the weighted points
        productsum=0;
        for iwin = 1:searchsize
            for jwin = 1:searchsize
                % Gaussian weight based on the intensity difference %Eq. 3.31
                weightp=exp(-(local_mean(y+jwin-halfsearch-1,x+iwin-halfsearch-1)-...
                    local_mean(y,x))^2/(2*st_dev*st_dev));
                %and add a weighted amount of the image information
                productsum=productsum+weightp*... %Eq 3.32
                    image(y+jwin-halfsearch-1,x+iwin-halfsearch-1);
                %and add up the weights
                weightsum=weightsum+weightp;
            end
        end
        filtered(y,x)=floor(productsum/weightsum);
    end
end
```

**Code 3.11 Non-Local Means Operator**

The effect of the non-local means operator compared with Gaussian averaging is shown in Figure 3.26. The image here Figure 3.26(a) is one to which synthetic noise has been added, and that is clearly seen to be removed by both operators. The function of the non-local means operator is to preserve regions of intensity and that is what is achieved (compare the wall of the hut in Figure 3.26(b) to that in Figure 3.26(c)). These images are not intended to be optimal, since optimisation depends on the application image and whether preservation of detail is more important than preservation of regions. This is implicit in the compromise between the choice of the size of the search region and the variance  $\sigma^2$ . A common choice for the window size,  $N$ , of the averaging operator is  $7 \times 7$  or  $9 \times 9$  to preserve local features and larger search sizes,  $M$ , to reduce noise. The concern on the selection of the range and its relationship with the noise has motivated approaches that adapt the range according to image content [Kervrann06]. The image here Figure 3.26(a) is one to which synthetic noise has been added, and that is clearly seen to be removed by both operators. Note that the image would originally contain noise without the addition of its synthesised form, but the results would be less easy to see. Performance analysis often depends on a chosen application.



Calling the operator non-local is actually rather misleading, as the originators later noted [Buades11] and a more appropriate name could have been semi-local means. Given that much of its popularity is due to performance, there is a natural interest in speeding the algorithm. One approach concentrates on implementation to improve speed [Dowson11] which, due to computational complexity, is needed when processing volumetric images.

### 3.5.4 Bilateral filtering

*Bilateral filtering* is a non-linear filter introduced by Tomasi and Manduchi [Tomasi98]. The filter is based on Gaussian averaging and prevents blurring across feature boundaries by decreasing the filter weight when the intensity difference is too large. By denoting Gaussian averaging as the convolution of the template in Equation 3.26 to form a new point  $\mathbf{G}_{x,y}$  as a weighted sum of image points within a template

$$\begin{aligned} \mathbf{G}_{x,y} &= \frac{1}{ks} \sum_{i \in N} \sum_{j \in N} \mathbf{O}_{x(i),y(j)} g(i, j, \sigma_d) \\ &= \frac{1}{\sum_{i \in N} \sum_{j \in N} g(i, j, \sigma_d)} \sum_{i \in N} \sum_{j \in N} \mathbf{O}_{x(i),y(j)} e^{-\frac{((x(i)-x)^2 + (y(j)-y)^2)}{2\sigma_d^2}} \end{aligned} \quad (3.33)$$

where  $\sigma_d$  is the standard deviation of this spatial (domain) operator,  $x(i), y(j)$  are again the coordinates of image points within the template,  $ks$  is the normalising coefficient, and the template is of size  $N \times N$ . The bilateral filtering process introduces another weighting based on the difference of image intensities. Inevitably, this is formed as a Gaussian (range) function in the manner of Equation 3.31

$$g(p, q, \sigma_r) = \frac{1}{2\pi\sigma_r^2} e^{-\frac{(p-q)^2}{2\sigma_r^2}} \quad (3.34)$$

where  $p, q$  are point intensities and  $\sigma_r$  is the standard deviation, controlling width. When the intensity difference is small then this function is large, and the operator preserves feature boundaries, preventing blurring. This is one of the advantages of the median operator and a disadvantage of the Gaussian smoothing operator. The bilateral filter is given by the combination of spatial averaging with the Gaussian weighted difference in brightness as

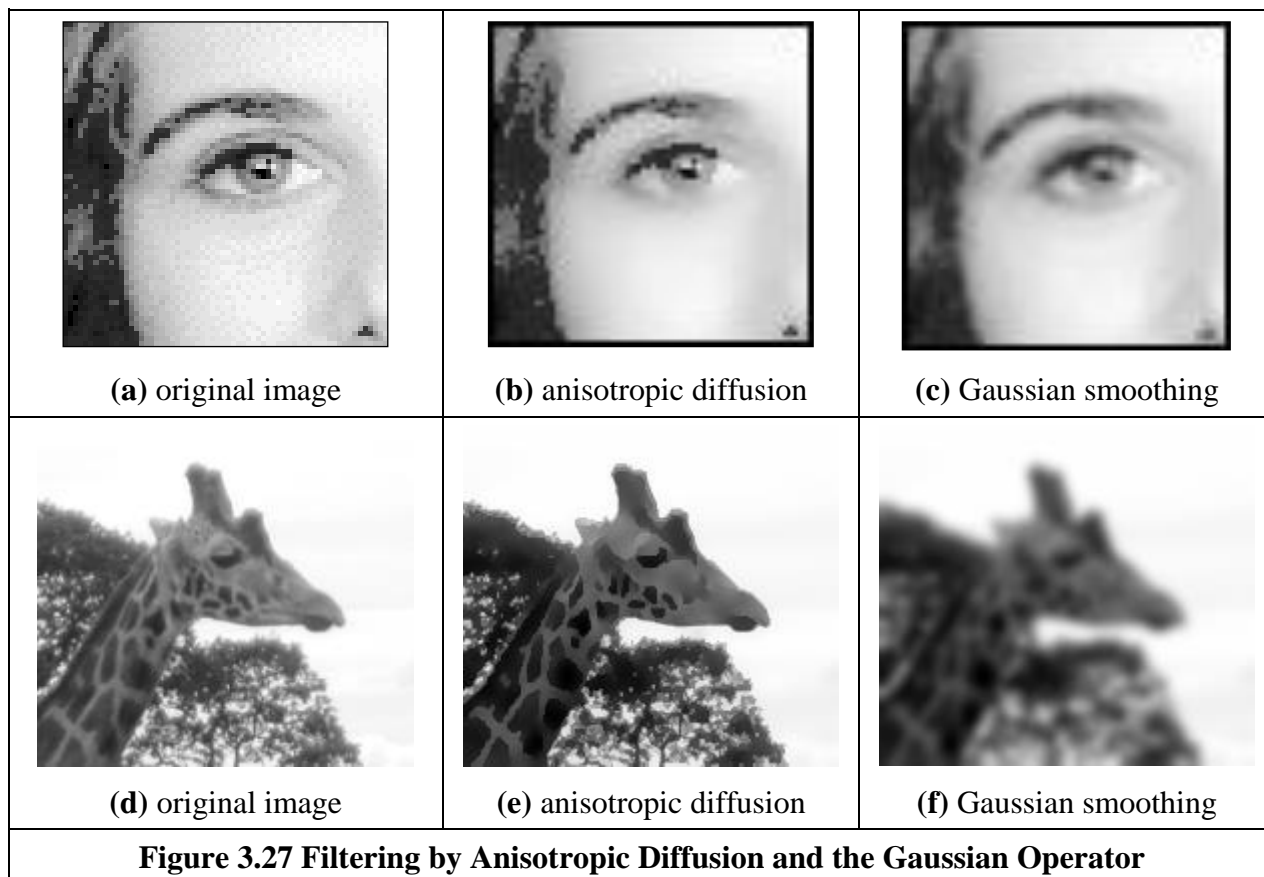
$$\begin{aligned} \mathbf{N}_{x,y} &= \frac{1}{kb} \sum_{i \in M} \sum_{j \in M} \mathbf{G}_{x(i),y(j)} g(\mathbf{O}_{x,y}, \mathbf{O}_{x(i),y(j)}, \sigma_r) \\ &= \frac{1}{kb} \sum_{i \in M} \sum_{j \in M} \mathbf{G}_{x(i),y(j)} e^{-\frac{(\mathbf{O}_{x,y} - \mathbf{O}_{x(i),y(j)})^2}{2\sigma_r^2}} \end{aligned} \quad (3.35)$$

where  $kb$  is the sum of the template coefficients. The parameters that must be chosen are the window sizes and the two values for standard deviation. Note that the operator reverts to a Gaussian operator when  $\sigma_r$  is large, and a range filter when  $\sigma_d$  is large. The difference from non-local means is that both areas and feature boundaries are preserved, though we shall not show the result here, partly as performance is similar to that of anisotropic diffusion, the following smoothing operator.

Optimised versions are available, which do not need parameter selection [Paris08, Weiss06]. Another method is based on the use of the frequency domain and uses linear filtering [Dabov07].

### 3.5.5 Anisotropic Diffusion

Another form of smoothing that preserves the boundaries of the image features in the smoothing process [Perona90]. The process is called *anisotropic diffusion*, by virtue of its basis. Its result is illustrated in Fig 3.27(b) where the feature boundaries (such as those of the eyebrows or the eyes) in the smoothed image are crisp and the skin is more matte in appearance. This implies that we are filtering within the features and not at their edges. By way of contrast, the Gaussian operator result in Figure 3.27(c) smooths not just the skin but also the boundaries (the eyebrows in particular seem quite blurred) giving a less pleasing and less useful result. Since we shall later use the boundary information to interpret the image, its preservation is of much interest. The example in the second row in Figure 3.27 illustrates how the filter is capable of maintaining fine structures. Notice how the branches of the trees in the background and the large scale texture detail are all clear.



As ever, there are some parameters to select to control the operation, so we shall consider the technique's basis so as to guide their selection. Further, it is computationally more complex than other filtering operators. The basis of anisotropic diffusion is however rather complex, especially here, and invokes concepts of low-level feature extraction which are covered in the next Chapter. One strategy you might use is to mark this page then go ahead and read Sections 4.1 and 4.2 and return here. Alternatively, you could just plough on, since that is exactly what we shall do. The complexity is because the process not only invokes low-level feature extraction (to preserve feature boundaries) but also as its basis actually invokes concepts of heat flow, as well as introducing the

concept of scale space. So it will certainly be a hard read for many, but comparison of Figure 3.27(b) with Figure 3.27(c) shows that it is well worth the effort.

The essential idea of scale space is that there is a multiscale representation of images, from low resolution (a coarsely sampled image) to high resolution (a finely sampled image). This is inherent in the sampling process where the coarse image is the structure and the higher resolution increases the level of detail. As such, we can derive a scale space set of images by convolving an original image with a Gaussian function,

$$\mathbf{P}_{x,y}(\sigma) = \mathbf{P}_{x,y}(0) * g(x, y, \sigma) \quad (3.36)$$

where  $\mathbf{P}_{x,y}(0)$  is the original image,  $g(x, y, \sigma)$  is the Gaussian template derived from Eqn. 3.26, and  $\mathbf{P}_{x,y}(\sigma)$  is the image at level  $\sigma$ . The coarser level corresponds to larger values of the standard deviation  $\sigma$ ; conversely the finer detail is given by smaller values. (Scale space will be considered again in Section 4.4.2 as it pervades the more modern operators). We have already seen that the larger values of  $\sigma$  reduce the detail and are then equivalent to an image at a coarser scale, so this is a different view of the same process. The difficult bit is that the family of images derived this way can equivalently be viewed as the solution of the heat equation

$$\frac{\partial \mathbf{P}}{\partial t} = \nabla \mathbf{P}_{x,y}(t) \quad (3.37)$$

where  $\nabla$  denotes del, the (directional) gradient operator from vector algebra, and with the initial condition that  $\mathbf{P}_0 = \mathbf{P}_{x,y}(0)$ . The heat equation itself describes the temperature  $T$  changing with time  $t$  as a function of the thermal diffusivity (related to conduction)  $\kappa$  as

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T \quad (3.38)$$

and in one dimensional form this is

$$\frac{\partial T}{\partial t} = \kappa \frac{\partial^2 T}{\partial x^2} \quad (3.39)$$

so the temperature measured along a line is a function of time, distance, the initial and boundary conditions and the properties of a material. The direct relation of this with image processing is clearly an enormous ouch! There are clear similarities between Eqn. 3.39 and Eqn. 3.37. This is the same functional form and allows for insight, analysis and parameter selection. The heat equation, Eqn. 3.37 is the anisotropic diffusion equation

$$\frac{\partial \mathbf{P}}{\partial t} = \nabla \cdot (c_{x,y}(t) \nabla \mathbf{P}_{x,y}(t)) \quad (3.40)$$

where  $\nabla \cdot$  is the divergence operator (which essentially measures how the density within a region changes), with diffusion coefficient  $c_{x,y}$ . The diffusion coefficient applies to the local change in the image  $\nabla \mathbf{P}_{x,y}(t)$  in different directions. If we have a lot of local change, we seek to retain it since the amount of change is the amount of boundary information. The diffusion coefficient indicates how much importance we give to local change: how much of it is retained. (The equation reduces to isotropic diffusion – Gaussian filtering - if the diffusivity is constant since  $\nabla c = 0$ .) There is no explicit solution to this equation. By approximating differentiation by differencing (this is explored more in Section 4.2) the rate of change of the image between time step  $t$  and time step  $t+1$  we have

$$\frac{\partial \mathbf{P}}{\partial t} = \mathbf{P}(t+1) - \mathbf{P}(t) \quad (3.41)$$

This implies we have an iterative solution, and for later consistency we shall denote the image  $\mathbf{P}$  at time step  $t+1$  as  $\mathbf{P}^{\langle t+1 \rangle} = \mathbf{P}(t+1)$  so we then have

$$\mathbf{P}^{\langle t+1 \rangle} - \mathbf{P}^{\langle t \rangle} = \nabla \cdot (c_{x,y}(t) \nabla \mathbf{P}^{\langle t \rangle}) \quad (3.42)$$

and again by approximation, using differences evaluated this time over the four compass directions North, South, East and West we have

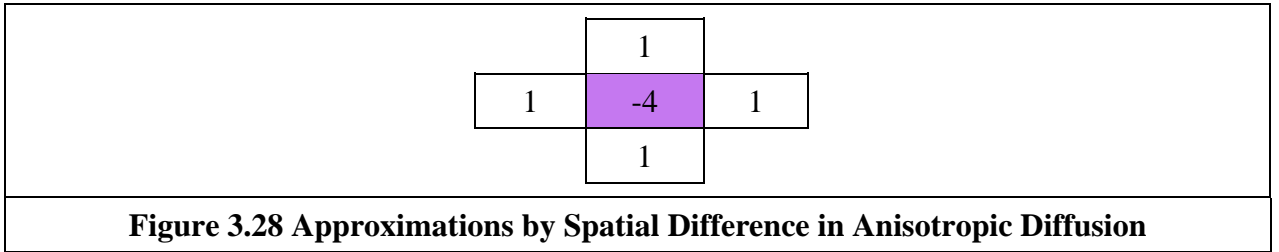
$$\nabla_N(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y-1} - \mathbf{P}_{x,y} \quad (3.43)$$

$$\nabla_S(\mathbf{P}_{x,y}) = \mathbf{P}_{x,y+1} - \mathbf{P}_{x,y} \quad (3.44)$$

$$\nabla_E(\mathbf{P}_{x,y}) = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} \quad (3.45)$$

$$\nabla_W(\mathbf{P}_{x,y}) = \mathbf{P}_{x-1,y} - \mathbf{P}_{x,y} \quad (3.46)$$

The template and weighting coefficients for these are shown in Figure 3.28.



When we use these as an approximation to the right hand side in Eqn. 3.42, we then have  $\nabla \cdot (c_{x,y}(t) \nabla \mathbf{P}^{\langle t \rangle}) = \lambda (cN_{x,y} \nabla_N(\mathbf{P}) + cS_{x,y} \nabla_S(\mathbf{P}) + cE_{x,y} \nabla_E(\mathbf{P}) + cW_{x,y} \nabla_W(\mathbf{P}))$  which gives

$$\mathbf{P}^{\langle t+1 \rangle} - \mathbf{P}^{\langle t \rangle} = \lambda (cN_{x,y} \nabla_N(\mathbf{P}) + cS_{x,y} \nabla_S(\mathbf{P}) + cE_{x,y} \nabla_E(\mathbf{P}) + cW_{x,y} \nabla_W(\mathbf{P})) \quad \left| \mathbf{P} = \mathbf{P}^{\langle t \rangle} \right. \quad (3.47)$$

where  $0 \leq \lambda \leq 1/4$  and where  $cN_{x,y}, cS_{x,y}, cE_{x,y}$  and  $cW_{x,y}$  denote the conduction coefficients in the four compass directions. By rearrangement of this we obtain the equation we shall use for the anisotropic diffusion operator

$$\mathbf{P}^{\langle t+1 \rangle} = \mathbf{P}^{\langle t \rangle} + \lambda (cN_{x,y} \nabla_N(\mathbf{P}) + cS_{x,y} \nabla_S(\mathbf{P}) + cE_{x,y} \nabla_E(\mathbf{P}) + cW_{x,y} \nabla_W(\mathbf{P})) \quad \left| \mathbf{P} = \mathbf{P}^{\langle t \rangle} \right. \quad (3.48)$$

This shows that the solution is iterative: images at one time step (denoted by  $\langle t+1 \rangle$ ) are computed from images at the previous time step (denoted  $\langle t \rangle$ ), given the initial condition that the first image is the original (noisy) image. Change (in time and in space) has been approximated as the difference between two adjacent points which gives the iterative equation and shows that the new image is formed by adding a controlled amount of the local change consistent with the main idea: that the smoothing process retains some of the boundary information.

We are not finished yet though, since we need to find values for  $cN_{x,y}, cS_{x,y}, cE_{x,y}$  and  $cW_{x,y}$ . These are chosen to be a function of the difference along the compass directions, so that the boundary (edge) information is preserved. In this way we seek a function that tends to zero with increase in the difference (an edge or boundary with greater contrast) so that diffusion does not take place across the boundaries, keeping the edge information. As such we seek

$$cN_{x,y} = g(\|\nabla_N(\mathbf{P})\|) \quad (3.49)$$

$$cS_{x,y} = g(\|\nabla_S(\mathbf{P})\|)$$



$$cE_{x,y} = g(\|\nabla_E(\mathbf{P})\|)$$

$$cW_{x,y} = g(\|\nabla_W(\mathbf{P})\|)$$

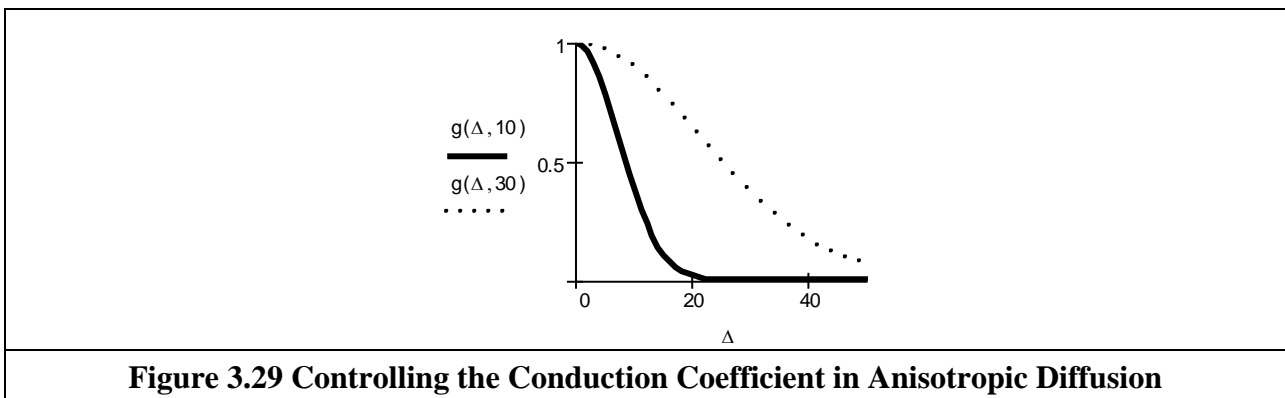
and one function that can achieve this is

$$g(x, k) = e^{-x^2/k^2} \tag{3.50}$$

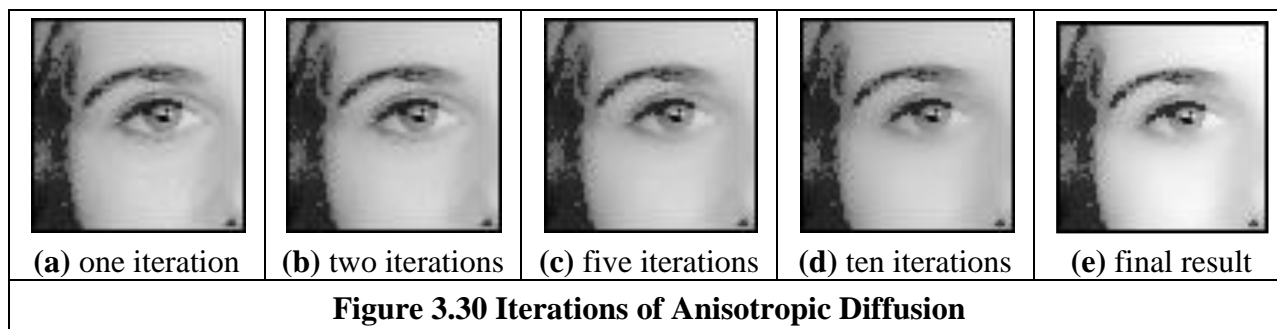
(There is potential confusion with using the same symbol as for the Gaussian function, Eqn. 3.26, but we have followed the original authors' presentation.) This function clearly has the desired properties since when the values of the differences  $\nabla$  are large the function  $g$  is very small, conversely when  $\nabla$  is small then  $g$  tends to unity.  $k$  is another parameter whose value we have to choose: it controls the rate at which the conduction coefficient decreases with increasing difference magnitude. The effect of this parameter is shown in Figure 3.29. Here, the solid line is for the smaller value of  $k$  and the dotted one is for a larger value. Evidently, a larger value of  $k$  means that the contribution of the difference reduces less than for a smaller value of  $k$ . In both cases, the resulting function is near unity for small differences and near zero for large differences, as required. An alternative to this is to use the function

$$g2(x, k) = \frac{1}{1 + x^2/k^2} \tag{3.51}$$

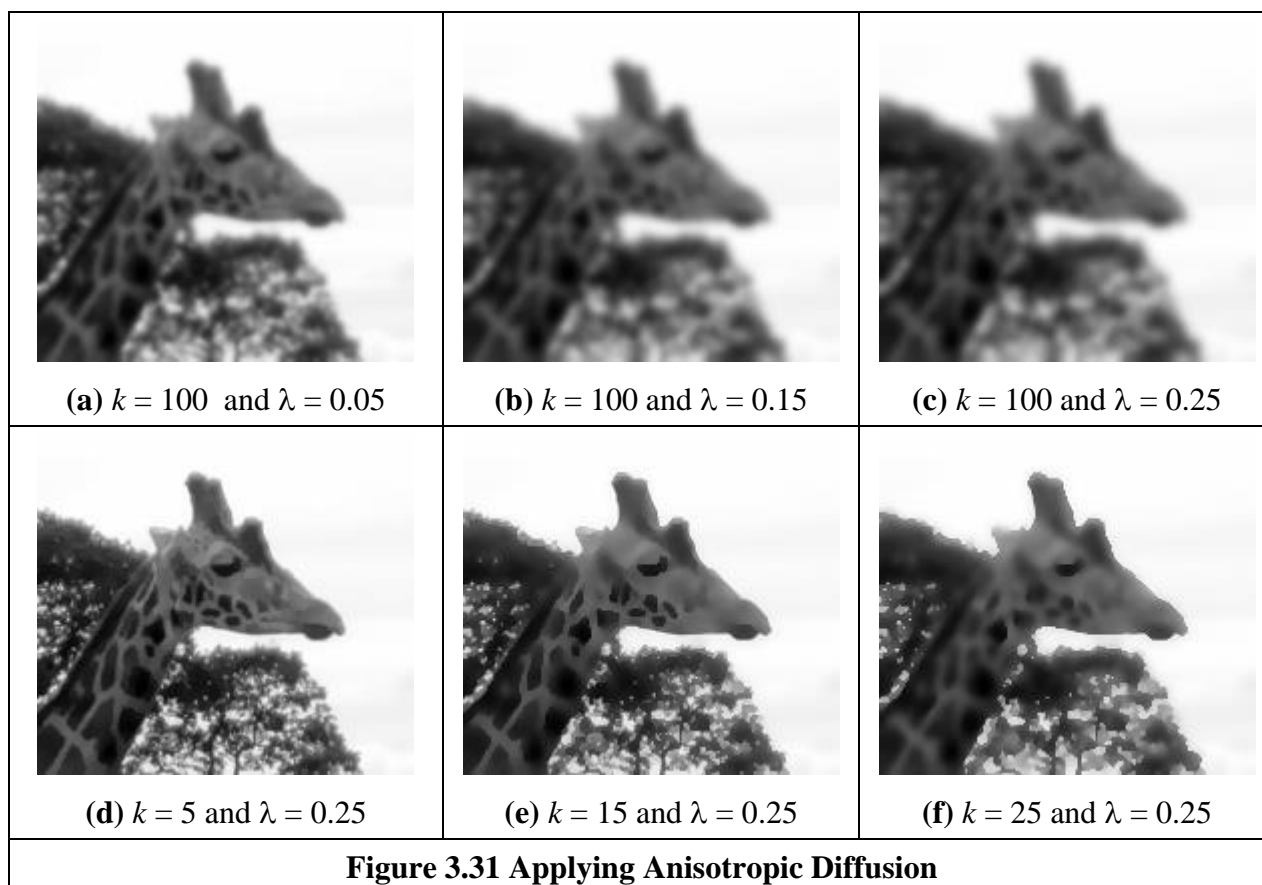
which has similar properties to the function in Eqn. 3.50.



This all looks rather complicated, so let's recap. First, we want to filter an image by retaining boundary points. These are retained according to the value of  $k$  chosen in Eqn. 3. 50. This function is operated in the four compass directions, to weight the brightness difference in each direction, Eqn. 3.49. These contribute to an iterative equation which calculates a new value for an image point by considering the contribution from its four neighbouring points, Eqn. 3.48. This needs choice of one parameter  $\lambda$ . Further, we need to choose the number of iterations for which calculation proceeds. For information, Figure 3.27(b) was calculated over 20 iterations and we need to use sufficient iterations to ensure that convergence has been achieved. Figure 3.30 shows how we approach this. Figure 3.30(a) is after a single iteration, (b) after 2, (c) after 5 and (d) after 10 and (e) after 20. Manifestly we could choose to reduce the number of iterations, accepting a different result – or even go further.



We also need to choose values for  $k$  and  $\lambda$ . By analogy,  $k$  is the conduction coefficient and low values preserve edges and high values allow diffusion (conduction) to occur – how much smoothing can take place. The two parameters are naturally inter-related, though  $\lambda$  largely controls the amount of smoothing. Given that low values of either parameter means that no filtering effect is observed, we can investigate their effect by setting one parameter to a high value and varying the other. In Figures 3.31(a)-(c) we use a high value of  $k$  which means that edges are not preserved and we can observe that different values of  $\lambda$  control the amount of smoothing. (A discussion of how this Gaussian filtering process is achieved can be inferred from Section 3.4.5.) Conversely, we can see how different values for  $k$  control the level of edge preservation in Figures 3.31(d)-(f) where some structures are not preserved for larger values of  $k$ .



By design, the result is akin with that of bilateral filtering (which it preceded) – indeed, the relationship has already been established [Barash02]. One of the major disadvantages is that anisotropic diffusion is an iterative process, the advantages include greater control of the filtering process. The original presentation of anisotropic diffusion [Perona90] is extremely lucid and well

worth a read if you consider selecting this technique. Naturally, it has greater detail on formulation and on analysis of results, than space here allows for (and is suitable at this stage). Amongst other papers on this topic, one [Black98] studied the choice of conduction coefficient leading to a function which preserves sharper edges and improves automatic termination. A more recent study has considered parameter and function choice [Tsotsios13]. As ever, with techniques that require much computation there have been approaches which speed implementation, or achieve similar performance faster (e.g. [Fischl99]).

Code 3.12 illustrates the implementation of the anisotropic process. The main loop contains the diffusion iterations. The iteration process uses a pair of images. The `outputImage` is the image computed in the previous iteration and `image` is the image updated in the current iteration. Each pixel value is updated according to the region defined by `kernelSize`. The implementation computes a weight for each pixel in the window according to Equation 3.50. This weight measures the differences in Equations 3.43 and is inversely proportional to the gradient or changes around the pixel. That is, when the pixel is close to an edge this value is low; and it will produce less smoothing. The weight is multiplied by `lambda` that controls the global smoothing. Notice that the weights are used to compute a simple average. In a more complete implementation the average could be replaced by a Gaussian operator that models the diffusion process more accurately. The examples in Figure 3.31 were obtained with this implementation.

```

for iteration in range(0, numIterations):
    for x,y in itertools.product(range(0, width), range(0, height)):
        image[y, x] = outputImage[y, x]

    for x,y in itertools.product(range(0, width), range(0, height)):
        sumWeights = 0;
        outputImage[y, x] = 0

        centrePixelValue = image[y, x]
        for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):
            posY, posX = y + wy - kernelCentre, x + wx - kernelCentre

            if posY > -1 and posY < height and posX > -1 and posX < width:
                # Weight according to gradient
                weight = exp(-pow((image[posY, posX]-centrePixelValue)/k, 2) );

                # Use lambda to weight the pixel value
                if posY != y and posX != x:
                    weight *= lamda

                sumWeights += weight
                outputImage[y, x] += weight * float(image[posY, posX])
        # Normalize
        if sumWeights > 0:
            outputImage[y, x] /= sumWeights

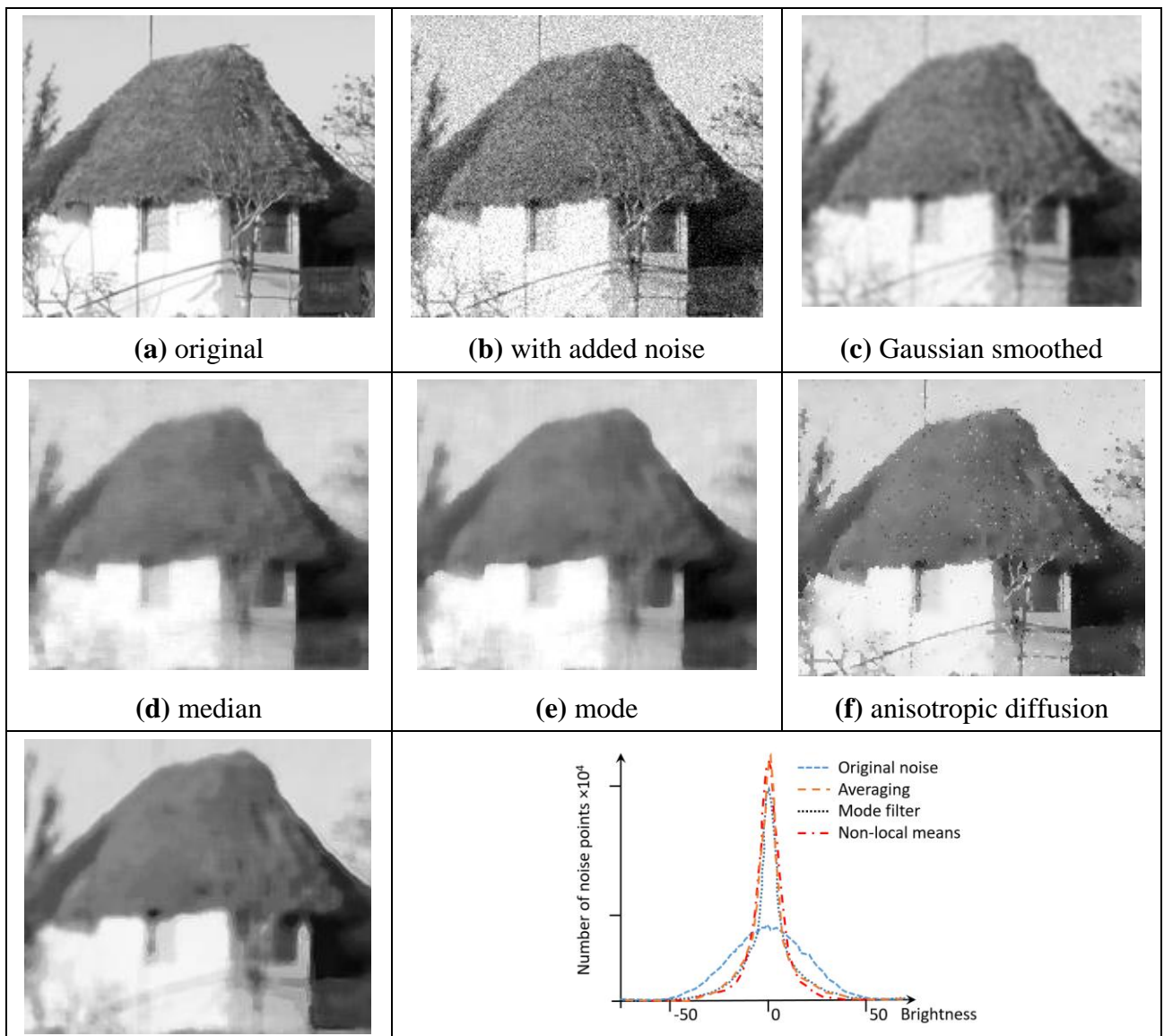
```

**Code 3.12 Anisotropic Diffusion Process**

### 3.5.6 Comparison of Smoothing Operators

A principled way to address performance is to add noise with known properties. This can be Gaussian noise for most operators, or black and white noise for the median operator. The results of some different image filtering operators are shown for comparison in Figure 3.32. (This is a detail – actually a restaurant called The Rock – as it is hard to see advantages in a whole image, as in Figure 3.26). The processed image is Figure 3.24(a) corrupted by added Gaussian noise, Figure 3.32(b). All operators are  $9 \times 9$  with parameters chosen for best performance. Naturally, ‘best’ is

subjective: is it best to retain feature boundaries (e.g. the roof edge or the foliage) or to retain local structure (e.g. the roof's thatching) – or just the local average (e.g. the wall)? That invariably depends on application. Figure 3.32(c), (d), (e), (f), (g) and (h) result from averaging (mean), Gaussian averaging, median, truncated median, anisotropic diffusion and non-local means, respectively. Each operator shows a different performance: the mean operator removes much noise but blurs feature boundaries; Gaussian averaging retains more features, but shows little advantage over direct averaging; the median operator retains some noise but with clear feature boundaries; whereas the truncated median removes more noise, but along with picture detail. Clearly, the increased size of the truncated median template, by the results in Figures 3.24(b) and (c), can offer improved performance. This is to be expected since by increasing the size of the truncated median template, we are essentially increasing the size of the distribution from which the mode is found. The effect of filtering can be assessed by studying the distribution of noise in the resulting image. The noise remaining after filtering can be separated by differencing the original image (without the noise added) from the resulting filtered image. Some histograms are shown in Figure 3.32 for the addition of noise, the effect of Gaussian averaging and non-local means. Clearly the noise is reduced (tends on average to zero), and such information can be helpful when optimising choice of the parameters used.



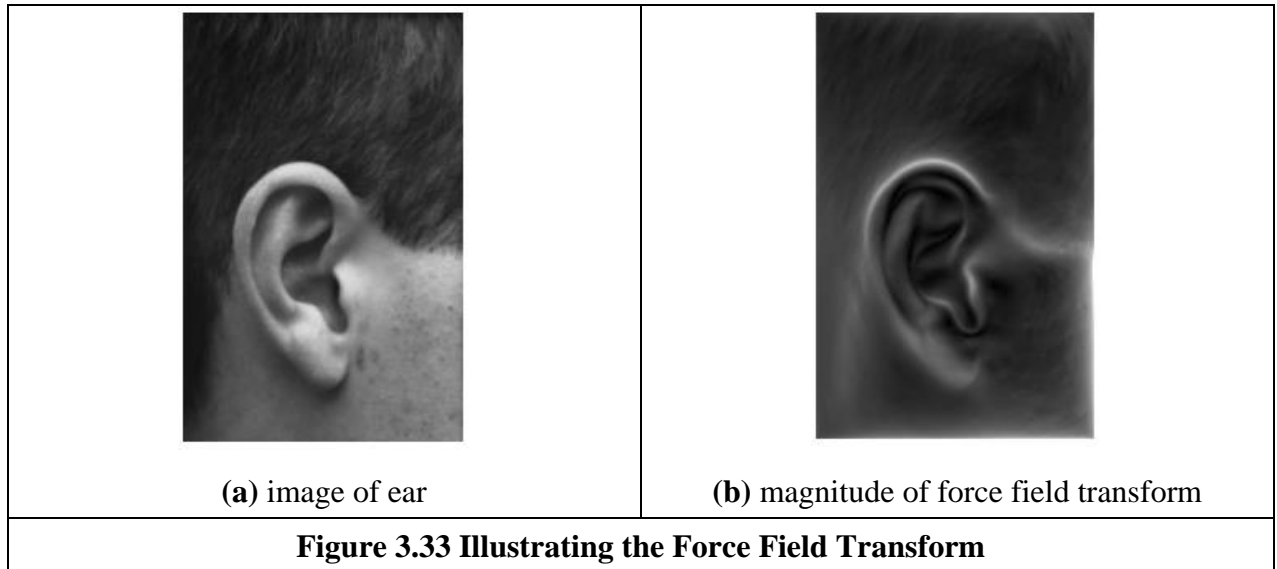
(g) non local means	(i) effect of filtering on noise
<b>Figure 3.32 Comparison of Filtering Operators</b>	

This performance evaluation has been for real images since that is more appealing, but when the noise is unknown we can only perform subjective analysis. Accordingly, better appraisal is based on the use of feature extraction. Boundaries are the low-level features studied in the next Chapter; shape is a high-level feature studied in Chapter 5. Also, we shall later use the filtering operators as a basis for finding objects which move in sequences of images, in Chapter 9.

Naturally, comparing the operators begs the question: when will denoising approaches be developed which are at the performance limit? Manifestly, the volume of research following development of a particular operator, say non local means, suggests that optimality is rarely achieved. One such study [Chatterjee10] concluded that “despite the phenomenal recent progress in the quality of denoising algorithms, some room for improvement still remains for a wide class of general images”. A later study [Levin11] showed, by modelling natural image statistics, that for small windows the state of the art denoising algorithms appeared to be approaching optimality with little room for improvement. A very recent approach concerns noise reduction and upsampling [Ham18]. There are now deep learning approaches for image quality improvement, e.g. [Zhang17] shows that discriminative model learning for image denoising has been attracting considerable attention due to its favourable performance. These will be introduced in Chapter 12. Denoising is not finished and perhaps may never be. However, having covered the major noise filtering operators, we shall finish with them and move on to operators that preserve selected features, since feature detection is the topic of the following Chapter.

### 3.5.7 Force Field Transform

There are of course many more image filtering operators; we have so far covered those that are amongst the most popular. There are others which offer alternative insight, sometimes developed in the context of a specific application. By way of example, Hurley developed a transform called the *force field transform* [Hurley02, Hurley05] which uses an analogy to gravitational force. The transform pretends that each pixel exerts a force on its neighbours which is inversely proportional to the square of the distance between them. This generates a force field where the net force at each point is the aggregate of the forces exerted by all the other pixels on a "unit test pixel" at that point. This very large scale summation affords very powerful averaging which reduces the effect of noise. The approach was developed in the context of ear biometrics, recognising people by their ears, which has unique advantage as a biometric in that the shape of people’s ears does not change with age, and of course – unlike a face - ears do not smile! The force field transform of an ear, Figure 3.33(a), is shown in Figure 3.33(b). Here, the averaging process is reflected in the reduction of the effects of hair. The transform itself has highlighted ear structures, especially the top of the ear and the lower ‘keyhole’ (the inter-tragic notch).



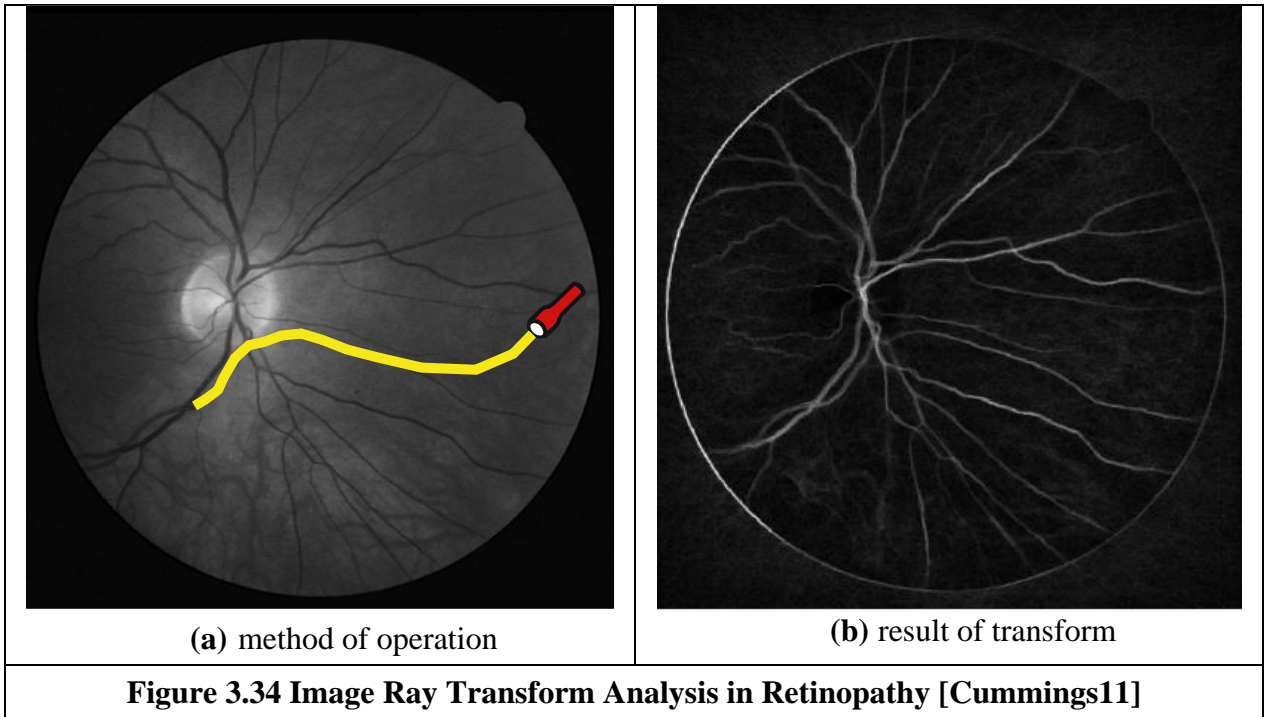
The image shown is actually the magnitude of the force field. The transform itself is a vector operation, and includes direction [Hurley02]. The transform is expressed as the calculation of the force  $\mathbf{F}$  between two points at positions  $\mathbf{r}_i$  and  $\mathbf{r}_j$  which is dependent on the value of a pixel at point  $\mathbf{r}_i$  as

$$\mathbf{F}_i(\mathbf{r}_j) = \mathbf{P}(\mathbf{r}_i) \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} \quad (3.52)$$

which assumes that the point  $\mathbf{r}_j$  is of unit “mass”. This is a directional force (which is why the inverse square law is expressed as the ratio of the difference to its magnitude cubed) and the magnitude and directional information has been exploited to determine an ear ‘signature’ by which people can be recognised. In application, Equation 3.52 can be used to define the coefficients of a template that is convolved with an image (implemented by the FFT to improve speed), as with many of the techniques that have been covered in this Chapter; an implementation is also given [Hurley02]. Note that this transform actually exposes low level features (the boundaries of the ears) which are the focus of the next Chapter. How we can determine shapes is a higher level process, and how the processes by which we infer or recognise identity from the low- and the high-level features will be covered in Chapter 12.

### 3.5.8 Image Ray Transform

The force field transform shows that there are other mathematical bases available for processing a whole image, as did the use of heat in the anisotropic diffusion operator. Another technique is called the *Image Ray Transform* (IRT) [Cummings11] and this uses an analogy to the propagation of light to transform an image to a new representation, with particular advantages.



As shown in Figure 3.34(a) the IRT operates in principle by initialising a torch in multiple positions and orientations and then tracing the path of the torch's beam. This analogises an image as a matrix of glass blocks, where the image brightness is proportional to the refractive index of each block. Essentially, the technique uses Snell's law which describes propagation of light. The angle of reflection  $\theta_r$  of a ray of light is calculated using Snell's law as

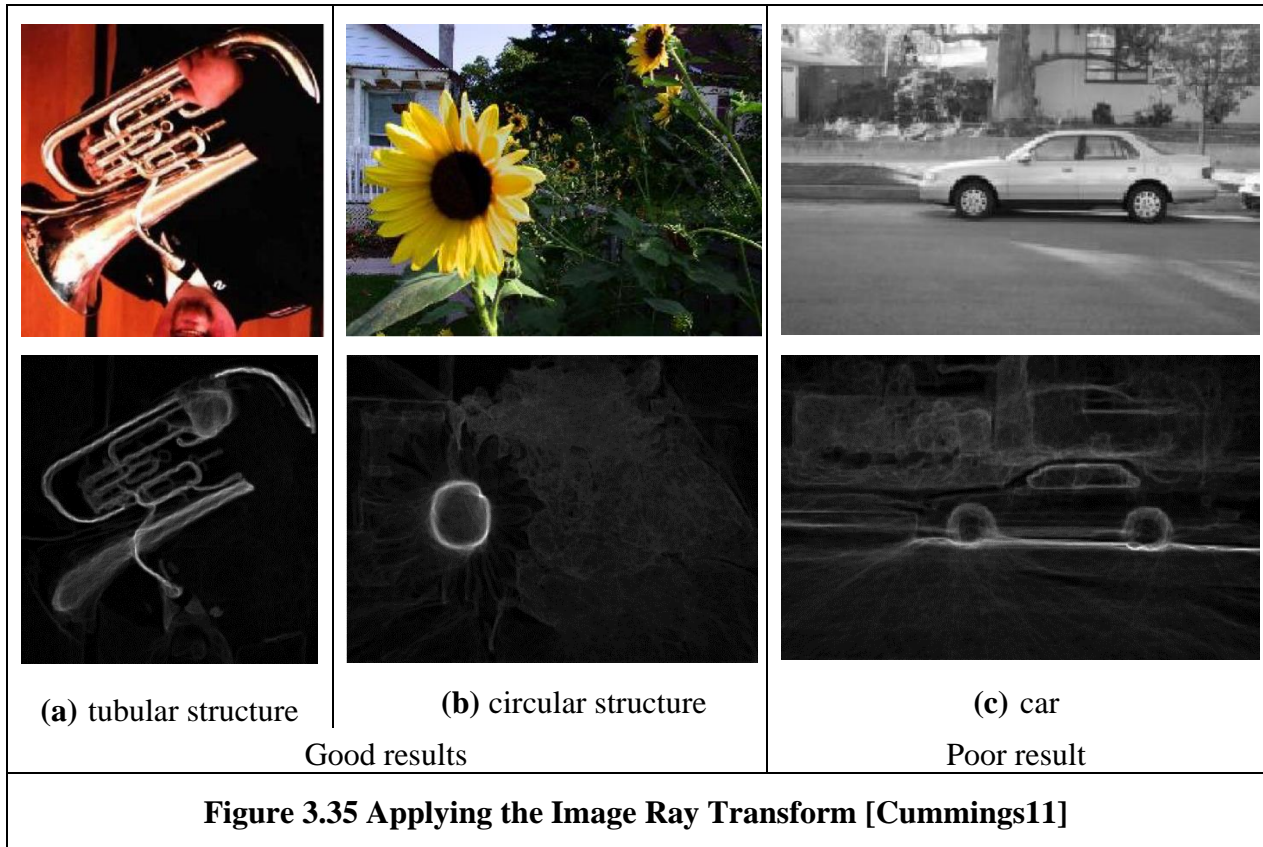
$$\frac{\sin \theta_i}{\sin \theta_r} = \frac{n_2}{n_1} \quad (3.53)$$

where  $\theta_i$  is the angle of an incident ray,  $n_1$  is the refractive index of the incident medium and is the  $n_2$  refractive index of the exit medium. Whether or not a ray is refracted or reflected depends on the critical angle that is calculated from the ratio of the refractive indices. The refractive index is calculated from the image intensity  $i$ , given a chosen maximum refractive index  $n_{\max}$ , as

$$n_i = 1 + \left( \frac{i}{255} \right) (n_{\max} - 1) \quad (3.54)$$

There are many other refinements necessary for the technique, such as ray length, number of rays, and other parameters used in the refractive index calculation.

A result of the IRT on the retina image of Figure 3.34(a) is shown in Figure 3.34(b). Essentially this shows the number of times beams crossed different points in the image. (It should be noted that diabetic retinopathy is of major interest in medicine, as it can show onset of diabetes. This image derives from the DRIVE dataset [Staal04] which is used within evaluation of techniques in this area.) In Figure 3.34(b), the transform has highlighted the vessels and not the optical disc (the bright part of the centre of the eye, where the vessels appear to converge). This changes the image representation in a way combining feature extraction with filtering.



By its nature, the IRT is suited to application in images that contain tubular or circular structures and the transform will emphasise (light is trapped in tubular structures, in a manner akin with optical fibres). This is shown in Figure 3.35(a) and (b) as well as a poor result in Figure 3.35(c). The poor result arises from an image where the two circular structures could be expected to be emphasised, but the resolution of the image is poor with respect to these two structures and the shadow of the car produces a stronger tubular structure, directing rays away from the wheels.

No technique is a panacea, and that should never be expected. There is a wide variety of processing operators possible, many more than have been shown here and in application one chooses the most appropriate. We have described the major operators and compared their performance, though only for demonstration purposes on image quality improvement. We shall move to techniques that explicitly locate low level image structures in the next Chapter. Before that we shall move to a complementary approach to the image processing that has been presented so far.

### 3.6 Mathematical Morphology

Mathematical morphology analyses images by using operators developed using set theory [Serra86, Soille13]. It was originally developed for binary images and was extended to include grey level data. The word morphology actually concerns shapes: in mathematical morphology we process images according to shape, by treating both as sets of points. In this way, morphological operators define *local transformations* that change pixel values that are represented as *sets*. The ways pixel values are changed is formalised by the definition of the *hit or miss transformation*.

In the hit or miss transform, an object represented by a set  $X$  is examined through a structural element represented by a set  $B$ . Different structuring elements are used to change the operations on the set  $X$ . The hit or miss transformation is defined as the point operator

$$X \otimes B = \{x \mid B_x^1 \subset X \cap B_x^2 \subset X^c\} \quad (3.55)$$





### 3.7 Further Reading

Many texts cover basic point and group operators in much detail, in particular some texts give many more examples, such as [Russ11], [Seul00] and there is a recent book intended to “introduce students to image processing and analysis” [Russ 2017]. Much material here is well established so the texts here are rather dated now, but note that basics never change – and there is always room for new techniques which offer improved performance. There is a relatively new text concentrating on image processing using Python [Chityala15]. Some of the more advanced texts include more coverage of low-level operators, such as Rosenfeld and Kak [Rosenfeld82] and Castleman [Castleman96]. There is MATLAB code for many of the low-level operations in this Chapter in [Gonzalez09]. For study of the effect of the median operator on image data, see [Bovik87]. Some techniques receive little treatment in the established literature, except for [Chan05] (with extensive coverage of noise filtering too). The Truncated Median Filter is covered again in Davies [Davies17]. Notwithstanding the discussion on more recent denoising operators at the end of Section 3.5.6, for further study of different statistical operators on ultrasound images, see [Evans96]. The concept of scale-space allows for considerably more refined analysis than is given here and we shall revisit it later. It was originally introduced by Witkin [Witkin83] and further developed by others including Koenderink [Koenderink84] (who also considers the heat equation). There is even series of conferences devoted to scale-space and morphology.

### 3.8 Chapter 3 References

- [Abd-Elmoniem 2002] Abd-Elmoniem, K. Z., Youssef, A. B., & Kadah, Y. M. Real-time speckle reduction and coherence enhancement in ultrasound imaging via nonlinear anisotropic diffusion. *IEEE Trans. on Biomed. Eng.*, **49**(9), 997-1014, 2002
- [Barash02] D. Barash, A Fundamental Relationship between Bilateral Filtering, Adaptive Smoothing and the Nonlinear Diffusion Equation, *IEEE Trans. on PAMI*, **24**(6), pp 844-849, 2002
- [Black98] Black, M. J., Sapiro, G., Marimont, D. H., and Meeger, D., Robust Anisotropic Diffusion, *IEEE Trans. on IP*, **7**(3), pp 421-432, 1998
- [Bovik87] Bovik A. C., Huang T. S., Munson D. C., The Effect Of Median Filtering on Edge Estimation and Detection, *IEEE Trans. on PAMI*, **9**(2), pp 181-194, 1987
- [Buades05] Buades, A., Coll, B., and Morel, J. M., A non-local algorithm for image denoising. *Proc. CVPR*, 2005, **2**, pp 60-65, 2005
- [Buades11] Buades, A., Coll, B., and Morel, J. M. Non-local means denoising. *Image Processing On Line*, **1**, pp 208-212, 2011
- [Campbell69] Campbell, J. D., *Edge Structure and the Representation of Pictures*, PhD Thesis, Univ. Missouri Columbia USA, 1969
- [Castleman96] Castleman, K. R., *Digital Image Processing*, Prentice Hall Inc., Englewood Cliffs N. J. USA, 1996
- [Chan05] Chan, T., and Shen, J., *Image Processing and Analysis: Variational, PDE, Wavelet, and Stochastic Methods*, Society for Industrial and Applied Mathematics, 2005

- [Chatterjee10] Chatterjee, P., and Milanfar, P., Is Denoising Dead? *IEEE Trans. on IP*, **19**(4), pp 895-911, 2010
- [Coupé09] Coupé, P., Hellier, P., Kervrann, C., and Barillot, C. Nonlocal Means-based Speckle Filtering for Ultrasound Images. *IEEE Trans. on IP*, **18**(10), pp 2221-2229, 2009
- [Chityala15] Chityala, R., and Pudipeddi, S., *Image Processing and Acquisition using Python*, CRC Press, Boca Raton FL USA, 2015
- [Cummings11] Cummings, A. H., Nixon, M. S., and Carter, J. N., The image ray transform for structural feature detection, *Pattern Recognition Letters*, **32**(15), pp 2053-2060, 2011
- [Dabov07] Dabov, K., Foi, A., Katkovnik, V., and Egiazarian, K., Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering, *IEEE Trans. on IP*, **16**(8), pp 2080-2095, 2007
- [Davies88] Davies, E. R., On the Noise Suppression Characteristics of the Median, Truncated Median and Mode Filters, *Pattern Recog. Lett.*, **7**(2), pp 87-97, 1988
- [Davies17] Davies, E. R., *Computer Vision: Principles, Algorithms, Applications, Learning*, Academic Press, 5<sup>th</sup> Edition, 2017
- [Dowson11] Dowson, N., & Salvado, O. Hashed nonlocal means for rapid image filtering. *IEEE Trans. on PAMI*, **33**(3), 485-499. 2011
- [Evans96] Evans, A. N., and Nixon M. S., Biased Motion-Adaptive Temporal Filtering for Speckle Reduction in Echocardiography, *IEEE Trans. Medical Imaging*, **15**(1), pp 39-50, 1996
- [Fischl99] Fischl, B., and Schwartz, E. L. Adaptive Nonlocal Filtering: a Fast Alternative to Anisotropic Diffusion for Image Enhancement, *IEEE Trans. on PAMI*, **21**(1) pp 42-48, 1999
- [Glasbey93] Glasbey, C. A., An Analysis of Histogram-Based Thresholding Algorithms, *CVGIP-Graphical Models and Image Processing*, **55** (6), pp 532-537, 1993
- [Gonzalez17] Gonzalez, R. C., and Woods, R. E., *Digital Image Processing*, 4<sup>th</sup> Edition, Pearson Education, 2017
- [Gonzalez09] Gonzalez, R. C., Woods, R. E., and Eddins, S. L., *Digital Image Processing using MATLAB*, Prentice Hall, 2<sup>nd</sup> Edition, 2009
- [Ham19] Ham, B., Cho, M., and Ponce, J., Robust guided image filtering using nonconvex potentials. *IEEE Trans. on PAMI*, **40**(1), pp 192-207, 2018
- [Hearn97] Hearn, D., and Baker, M. P., *Computer Graphics C Version*, 2nd Edition, Prentice Hall, Inc., Upper Saddle River, NJ USA, 1997
- [Hodgson85] Hodgson, R. M., Bailey, D. G., Naylor, M. J., Ng A., and McNeill S. J., Properties, Implementations and Applications of Rank Filters, *Image and Vision Computing*, **3**(1), pp 3-14, 1985
- [Huang79] Huang, T., Yang, G., and Tang, G., A Fast Two-dimensional Median Filtering Algorithm, *IEEE Trans. on ASSP*, **27**(1), pp 13-18, 1979
- [Hurley02] Hurley, D. J., Nixon, M. S. and Carter, J. N., Force Field Energy Functionals for Image Feature Extraction, *Image and Vision Computing*, **20**, pp 311-317, 2002
- [Hurley05] Hurley, D. J., Nixon, M. S. and Carter, J. N., Force Field Feature Extraction for Ear Biometrics, *Computer Vision and Image Understanding*, **98**(3), pp 491-512, 2005

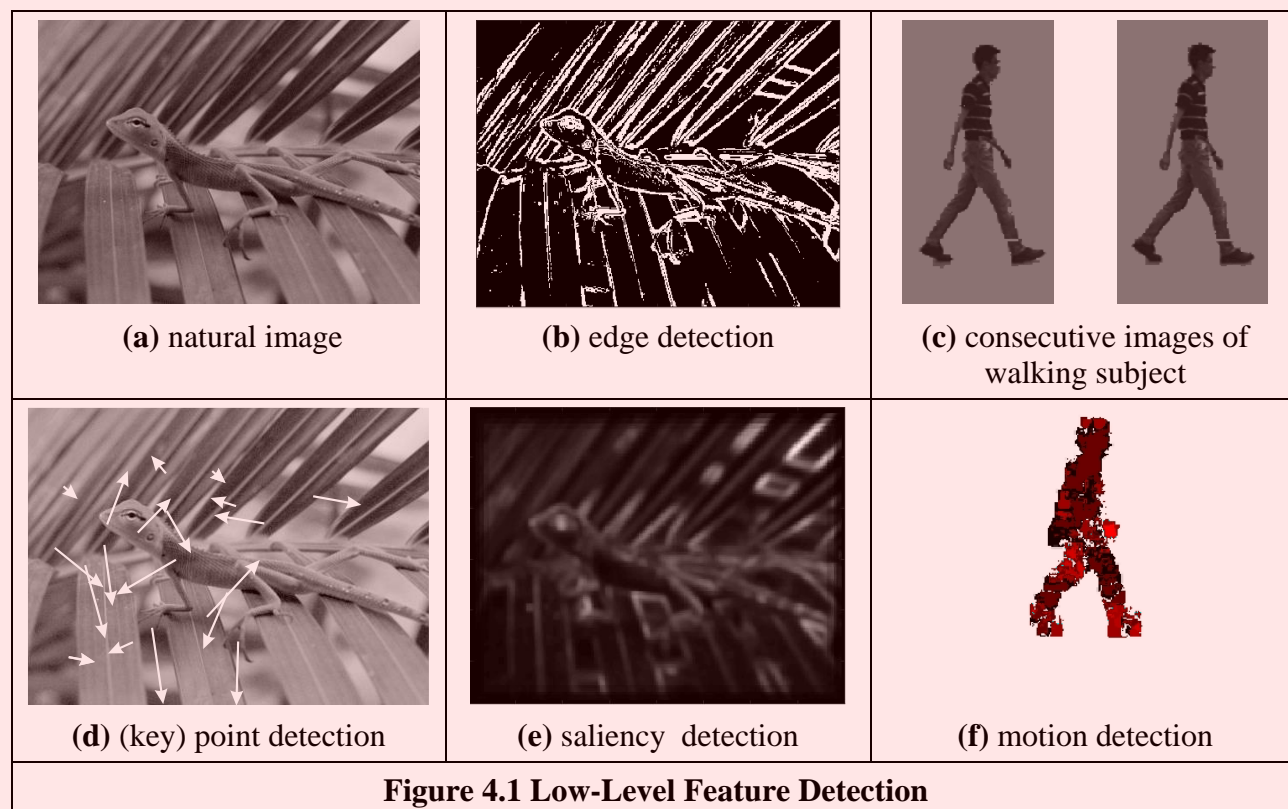
- [Kervrann06] Kervrann, C., and Boulanger, J. Optimal spatial adaptation for patch-based image denoising. *IEEE Trans. on IP*, **15**(10), 2866-2878, 2006
- [Koenderink84] Koenderink, J., The Structure of Images, *Biological Cybernetics*, **50**, pp 363-370, 1984
- [Lee90] Lee, S. A., Chung, S. Y. and Park, R. H., A Comparative Performance Study of Several Global Thresholding Techniques for Segmentation, *CVGIP*, **52**, pp 171-190, 1990
- [Levin11] Levin, A., and Nadler, B., Natural image denoising: optimality and inherent bounds, Proc. *CVPR*, 2011
- [Loupas87] Loupas, T., and McDicken, W. N., Noise Reduction in Ultrasound Images by Digital Filtering, *British Journal of Radiology*, **60**, pp 389-392, 1987
- [Montiel95] Montiel, M. E., Aguado, A. S., Garza, M., and Alarcón, J., Image Manipulation using M-filters in a Pyramidal Computer Model, *IEEE Trans. on PAMI*, **17**(11), pp 1110-1115, 1995.
- [OpenCV-TM] OpenCV Template Operator  
<https://github.com/opencv/opencv/blob/master/modules/imgproc/src/filter.cpp>
- [Otsu79] Otsu, N., A Threshold Selection Method from Gray-Level Histograms, *IEEE Trans. on SMC*, **9**(1), pp 62-66, 1979
- [Paris08] Paris, S., and Durand, F., A Fast Approximation of the Bilateral Filter Using a Signal Processing Approach, *International J. Computer Vision*, **81**(1), pp 24 – 52, 2008
- [Perona90] Perona, P., and Malik, J., Scale-Space and Edge Detection using Anisotropic Diffusion, *IEEE Trans. on PAMI*, **17**(7), pp 629-639, 1990
- Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ... and Zuiderveld, K. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, **39**(3), 355-368, 1987
- [Rosenfeld82] Rosenfeld, A and Kak A. C., *Digital Picture Processing*, 2<sup>nd</sup> Edition, Vols. 1 and 2, Academic Press Inc., Orlando FL USA, 1982
- [Rosin01] Rosin, P. L., Unimodal Thresholding, *Pattern Recognition*, **34**(11), pp 2083-2096, 2001
- [Russ11] Russ, J. C., *The Image Processing Handbook*, 6<sup>th</sup> Edition, CRC Press (IEEE Press), Boca Raton FL USA, 2011
- [Russ17] Russ, J. C., and Russ, J. C. *Introduction to Image Processing and Analysis*, CRC Press, Boca Raton FL USA, 2017
- [Seul00] Seul, M., O’Gorman, L., and Sammon, M. J., *Practical Algorithms for Image Analysis: Descriptions, Examples, and Code*, Cambridge University Press, Cambridge UK, 2000
- [Sahoo88] Sahoo, P. K., Soltani, S. Wong, A. K. C., and Chen, Y. C., Survey of Thresholding Techniques, *CVGIP*, **41**(2), pp 233-260, 1988
- [Serra86] Serra, J., Introduction to Mathematical Morphology, *Computer Vision, Graphics and Image Processing*, **35**, pp 283-305, 1986
- [Soille13] Soille, P., *Morphological Image Analysis: Principles and Applications*, Springer Science & Business Media, Heidelberg, Germany, 2013

- [Shankar86] Shankar, P. M., Speckle Reduction in Ultrasound B Scans using Weighted Averaging in Spatial Compounding, *IEEE Trans. on Ultrasonics, Ferroelectrics and Frequency Control*, **33**(6), pp754-758, 1986
- [Sternberg86] Sternberg, S. R., Gray Scale Morphology, *Computer Vision, Graphics and Image Processing*, **35**, pp 333-355, 1986
- [Staal04] Staal, J., Abramoff, M. Niemeijer, M.,Viergever, M., van Ginneken, B., Ridge based vessel segmentation in color images of the retina, *IEEE Trans. Med. Imag.*, **23**, pp 501-509, 2004
- [Tomasi98] C. Tomasi, and R. Manduchi, Bilateral filtering for Gray and Color images, Proc. *ICCV 1998*, pp 839-846, Bombay, India, 1998
- [Trier95] Trier, O. D., and Jain, A. K., Goal-Directed Evaluation of Image Binarisation Methods, *IEEE Trans. on PAMI*, **17**(12), pp 1191-1201, 1995
- [Tsiotsios13] Tsiotsios, C., and Petrou, M. On the choice of the parameters for anisotropic diffusion in image processing. *Pattern Recog.*, **46**(5), 1369-1381, 2013
- [Weiss06] Weiss, B., Fast median and bilateral filtering, *Proc. ACM SIGGRAPH 2006*, pp 519-526, 2006
- [Witkin83] Witkin, A., Scale-Space Filtering: a New Approach to Multi-Scale Description, *Proc. Int. Joint Conf. Artificial Intelligence*, pp 1019-1021, 1983
- [Zhang17] Zhang, K., Zuo, K., Chen, Y., Meng, D., Beyond a Gaussian denoiser: residual learning of deep CNN for image denoising, *IEEE Trans. IP*, **26**(7), pp 3142-3155, 2017

## 4 Low-Level Feature Extraction (including Edge Detection)

### 4.1 Overview

We shall define *low-level features* to be those basic features that can be extracted automatically from an image without any shape information (information about spatial relationships). As such, thresholding is actually a form of low-level feature extraction performed as a point operation. Naturally, all of these approaches can be used in high-level feature extraction, where we find shapes in images. It is well known that we can recognise people from caricaturists' portraits. That is the first low level feature we shall encounter. It is called *edge detection* and it aims to produce a line drawing, like Figure 4.1(b), something akin to a caricaturist's sketch though without the exaggeration a caricaturist would imbue. There are very basic techniques and more advanced ones and we shall look at some of the most popular approaches. The first order detectors are equivalent to first order differentiation and, naturally, the second order edge detection operators are equivalent to a one-higher level of differentiation. An alternative form of edge detection is called phase congruency and we shall again see the frequency domain used to aid analysis, this time for low-level feature extraction.



We shall also consider corner detection which can be thought of as detecting those points where lines bend very sharply with high curvature, such as the lizard's head in Figures 4.1(a) and (b). These are another low-level features that again can be extracted automatically from the image. These are largely techniques for localised feature extraction, in this case the curvature Figure 4.1(d), and the more modern approaches extend to the detection of localised regions or patches of interest, Figure 4.1(e). Finally, we shall investigate a technique that describes motion, called optical flow. This is illustrated in Figures 4.1(c) and (f) with the optical flow from images of a walking man: the bits that are moving fastest are the brightest points, like the hands and the feet. All of these can provide a set of points, albeit points with different properties, but all are suitable for grouping for

shape extraction. Consider a square box moving through a sequence of images. The edges are the perimeter of the box; the corners are the apices; the flow is how the box moves. All these can be collected together to find the moving box. We shall start with the edge detection techniques, with the first order operators which accords with the chronology of development. The first order techniques date back by more than 30 years.

Main topic	Sub topics	Main points
First order edge detection	What is an edge and how we detect it. The equivalence of operators to first order differentiation and the insight this brings. The need for filtering and more sophisticated first order operators.	Difference operation; <i>Roberts Cross</i> , <i>Smoothing</i> , <i>Prewitt</i> , <i>Sobel</i> , <i>Canny</i> . Basis of the operators and frequency domain analysis.
Second order edge detection	Relationship between first and second order differencing operations. The basis of a second order operator. The need to include filtering and better operations.	Second order differencing; <i>Laplacian</i> , <i>zero-crossing detection</i> ; <i>Marr-Hildreth</i> , <i>Laplacian of Gaussian</i> , <i>Difference of Gaussians</i> . <i>Scale space</i> .
Other edge operators	Alternative approaches and performance aspects. Comparing different operators.	Other noise models; <i>Spacek</i> . Other edge models; <i>Petrou</i> and <i>Susan</i> .
Phase congruency	Phase for feature extraction; inverse Fourier transform;. Alternative form of edge and feature detection.	Frequency domain analysis; detecting a range of features; photometric invariance, wavelets.
Localised feature extraction	Finding localised low-level features; extension from curvature to patches. Nature of curvature and computation from: edge information; by change in intensity; and by correlation. Motivation of patch detection and principles of modern approaches. Saliency and saliency operators.	<i>Planar curvature</i> ; corners. Curvature estimation by: change in <i>edge direction</i> ; intensity change; <i>Harris</i> corner detector. Patch based detectors; scale space. <i>SIFT</i> , <i>SURF</i> , <i>FAST</i> , <i>ORB</i> , <i>FREAK</i> , <i>Brightness Clustering</i> . Saliency operators. <i>context aware saliency</i> .
Optical flow Estimation	Movement and the nature of optical flow. Estimating the optical flow by a differential approach. Need for other approaches (inc. matching regions).	Detection by differencing. <i>Optical flow</i> ; <i>velocity</i> ; <i>aperture problem</i> ; smoothness constraint. <i>Differential approach</i> ; Horn and Schunk method; <i>correlation</i> ; Extensions: <i>Deepflow</i> ; <i>Epicflow</i> and <i>acceleration</i> . <i>Evaluation</i> of optical flow.

**Table 4.1 Overview of Chapter 4**

## 4.2 Edge Detection

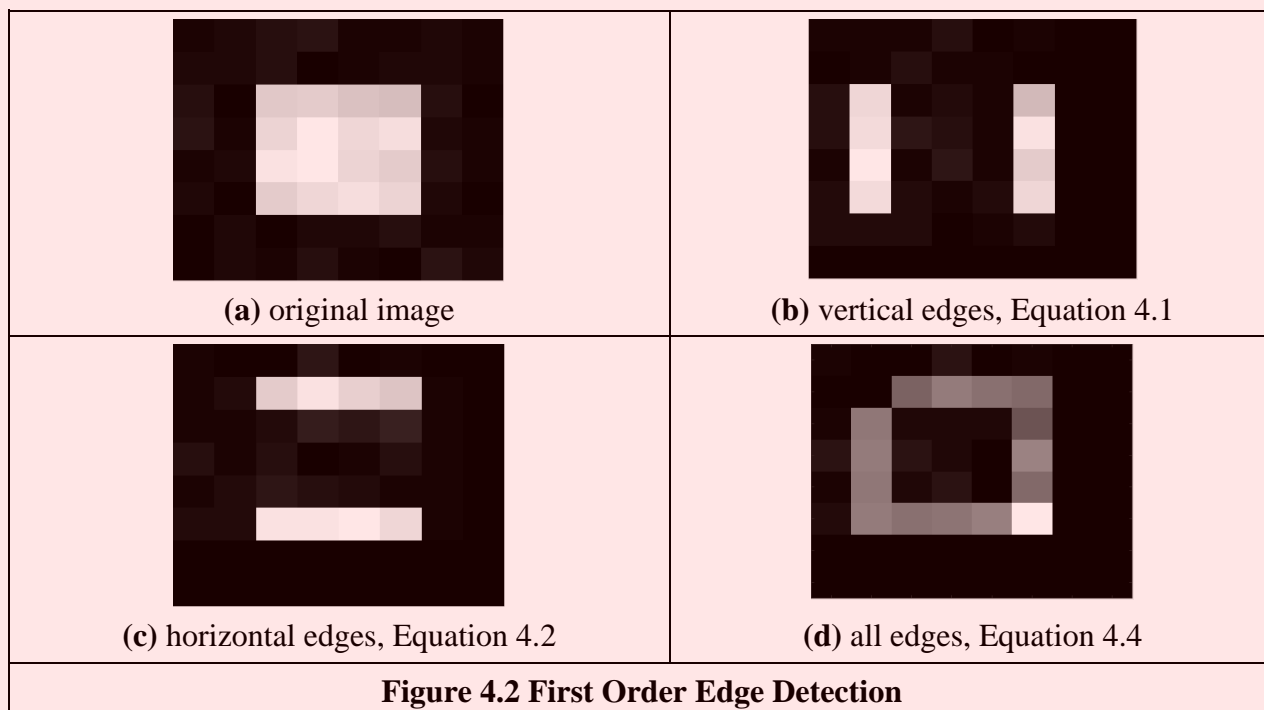
### 4.2.1 First Order Edge Detection Operators

#### 4.2.1.1 Basic Operators

Many approaches to image interpretation are based on edges, since analysis based on edge detection is insensitive to change in the overall illumination level. *Edge detection* highlights *image contrast*. Detecting contrast, which is difference in intensity, can emphasise the boundaries of features within an image, since this is where image contrast occurs. This is, naturally, how human vision can perceive the perimeter of an object, since the object is of different intensity to its surroundings. Essentially, the boundary of an object is a step-change in the intensity levels. The edge is at the position of the step change. To detect the edge position we can use *first order* differentiation since this emphasises change; first order differentiation gives no response when applied to signals that do not change. The first edge detection operators to be studied here are group operators which aim to deliver an output which approximates the result of first order differentiation.

A change in intensity can be revealed by differencing adjacent points. Differencing horizontally adjacent points will detect vertical changes in intensity and is often called a *horizontal edge-detector* by virtue of its action. A horizontal operator will not show up horizontal changes in intensity since the difference is zero. (This is the form of edge detection used within the anisotropic diffusion smoothing operator in the previous Chapter.) When applied to an image  $\mathbf{P}$  the action of the horizontal edge-detector forms the difference between two horizontally-adjacent points, as such detecting the vertical edges,  $\mathbf{E}_x$ , as:

$$\mathbf{E}_{x,y} = \left| \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} \right| \quad \forall x \in 1, N-1; y \in 1, N \quad (4.1)$$



In order to detect horizontal edges we need a *vertical edge-detector* which differences vertically adjacent points. This will determine horizontal intensity changes, but not vertical ones so the vertical edge-detector detects the horizontal edges,  $\mathbf{E}_y$ , according to:



$$\mathbf{E}y_{x,y} = \left| \mathbf{P}_{x,y} - \mathbf{P}_{x,y+1} \right| \quad \forall x \in 1, N; y \in 1, N-1 \quad (4.2)$$

Figures 4.2(b) and (c) show the application of the vertical and horizontal operators to the synthesised image of the square in Figure 4.2(a). The left hand vertical edge in Figure 4.2(b) appears to be beside the square by virtue of the forward differencing process. Likewise, the upper edge in Figure 4.2(c) appears above the original square.

The combination of the horizontal and vertical detectors defines an operator  $\mathbf{E}$  that can detect vertical and horizontal edges together. That is

$$\mathbf{E}_{x,y} = \left| \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x,y+1} \right| \quad \forall x, y \in 1, N-1 \quad (4.3)$$

which gives:

$$\mathbf{E}_{x,y} = \left| 2 \times \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y+1} \right| \quad \forall x, y \in 1, N-1 \quad (4.4)$$

Equation 4.4 gives the coefficients of a differencing template which can be convolved with an image to detect all the edge points, such as those shown in Figure 4.2(d). As in the previous Chapter, the current point of a template's operation (the position of the point we are computing a new value for) is shaded. The template shows only the weighting coefficients and not the modulus operation. Note that the bright point in the upper left corner of the edges of the square in Figure 4.2(d) is much brighter than the other points. This is because it is the only point to be detected as an edge by both the vertical and the horizontal operators and is therefore much brighter than the other edge points. In contrast, the lower right corner point is detected by neither operator and so does not appear in the final image.

2	-1
-1	0

**Figure 4.3 Template for First Order Difference**

The template in Figure 4.3 is convolved with the image to detect edges. The direct implementation of this operator, i.e. using Equation 4.4 rather than template convolution, is given in Code 4.1. Naturally, template convolution could be used, but it is unnecessarily complex in this case.

```
function edge = basic_difference(image)
for x = 1:cols-2 %address all columns except border
  for y = 1:rows-2 %address all rows except border
    edge(y,x)=abs(2*image(y,x)-image(y+1,x)-image(y,x+1)); % Eq. 4.4
  end
end
```

**Code 4.1 First Order Edge Detection**

*Uniform thresholding* (Section 3.3.4) can be used to select the brightest edge points. The threshold level controls the number of selected points; too high a level can select too few points, whereas too low a level can select too much noise. Often, the threshold level is chosen by experience or by experiment, but it can be determined automatically by considering edge data [Venkatesh95], or empirically [Haddon88]. For the moment, let us concentrate on the development of edge detection operators, rather than on their application.

4.2.1.2 Analysis of the Basic Operators

Taylor series analysis reveals that differencing adjacent points provides an estimate of the first order derivative at a point. If the difference is taken between points separated by  $\Delta x$  then by Taylor expansion for  $f(x + \Delta x)$  we obtain:

$$f(x + \Delta x) = f(x) + \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) + O(\Delta x^3) \tag{4.5}$$

By rearrangement, the first order derivative  $f'(x)$  is:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} - O(\Delta x) \tag{4.6}$$

This shows that the difference between adjacent points is an estimate of the first order derivative, with error  $O(\Delta x)$ . This error depends on the size of the interval  $\Delta x$  and on the complexity of the curve. When  $\Delta x$  is large this error can be significant. The error is also large when the high order derivatives take large values. In practice, the close sampling of image pixels and the reduced high frequency content make this approximation adequate. However, the error can be reduced by spacing the differenced points by one pixel. This is equivalent to computing the first order difference delivered by Equation 4.1 at two adjacent points, as a new horizontal difference **E<sub>xx</sub>** where

$$\mathbf{E}_{xx}_{x,y} = \mathbf{E}_{x_{x+1,y}} + \mathbf{E}_{x_{x,y}} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x-1,y} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y} \tag{4.7}$$

This is equivalent to incorporating spacing to detect the edges **E<sub>xx</sub>** by:

$$\mathbf{E}_{xx}_{x,y} = |\mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y}| \quad \forall x \in 2, N-1; y \in 1, N \tag{4.8}$$

To analyse this, again by Taylor series, we expand  $f(x - \Delta x)$  as:

$$f(x - \Delta x) = f(x) - \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) - O(\Delta x^3) \tag{4.9}$$

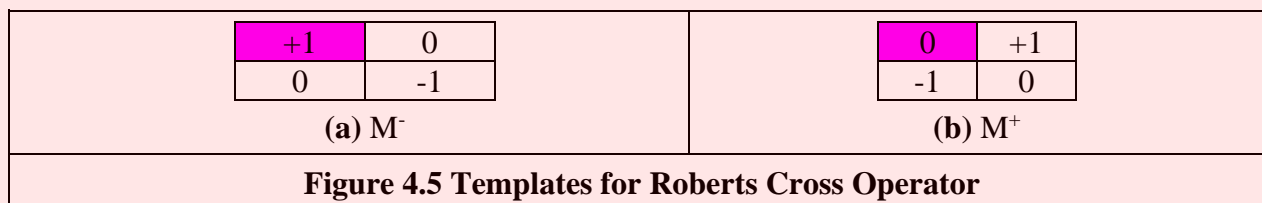
By differencing Equation 4.9 from Equation 4.5, we obtain the first order derivative as:

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} - O(\Delta x^2) \tag{4.10}$$

Equation 4.10 suggests that the estimate of the first order difference is now the difference between points separated by one pixel, with error  $O(\Delta x^2)$ . If  $\Delta x < 1$ , this error is clearly smaller than the error associated with differencing adjacent pixels, in Equation 4.6. Again, averaging has reduced noise, or error. The template for a horizontal edge-detection operator is given in Figure 4.4(a). This template gives the vertical edges detected at its centre pixel. A transposed version of the template gives a vertical edge-detection operator, Figure 4.4(b).

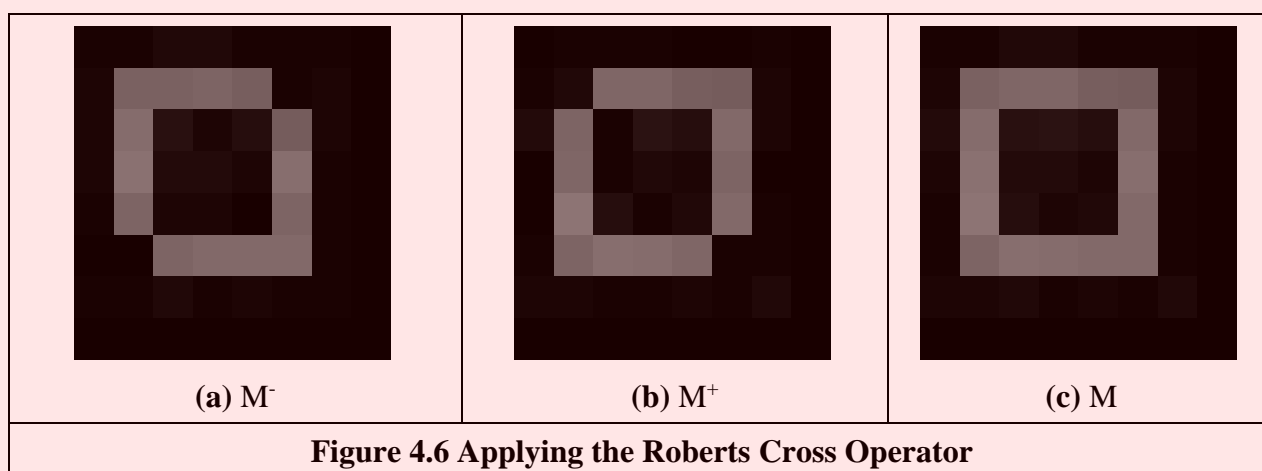


The *Roberts cross operator* [Roberts65] was one of the earliest edge detection operators. It implements a version of basic first order edge detection and uses two templates which difference pixel values in a diagonal manner, as opposed to along the axes' directions. The two templates are called  $M^+$  and  $M^-$  and are given in Figure 4.5.



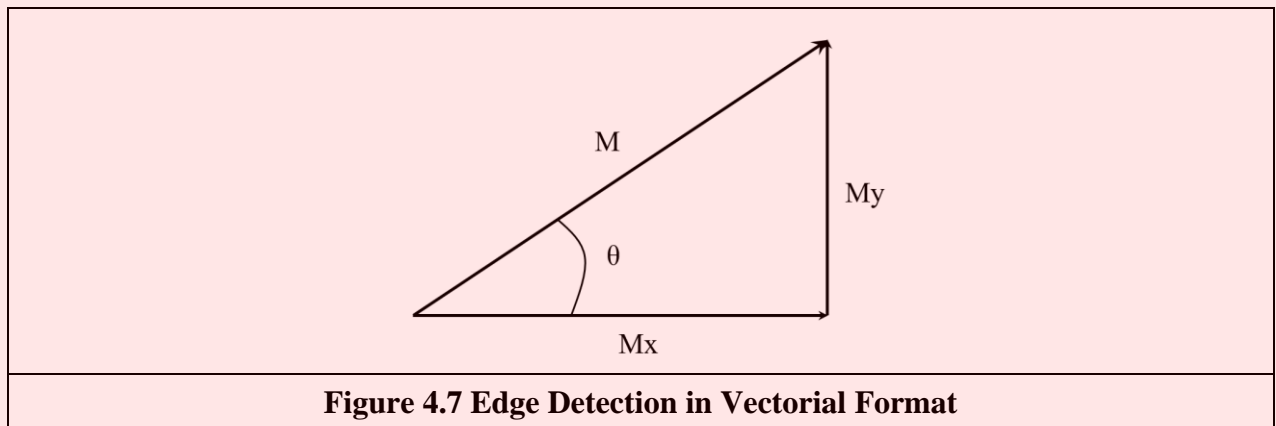
In implementation, the maximum value delivered by application of these templates is stored as the value of the edge at that point. The edge point  $E_{x,y}$  is then the maximum of the two values derived by convolving the two templates at an image point  $P_{x,y}$ :

$$E_{x,y} = \max \left\{ |M^+ * P_{x,y}|, |M^- * P_{x,y}| \right\} \quad \forall x, y \in 1, N-1 \quad (4.11)$$



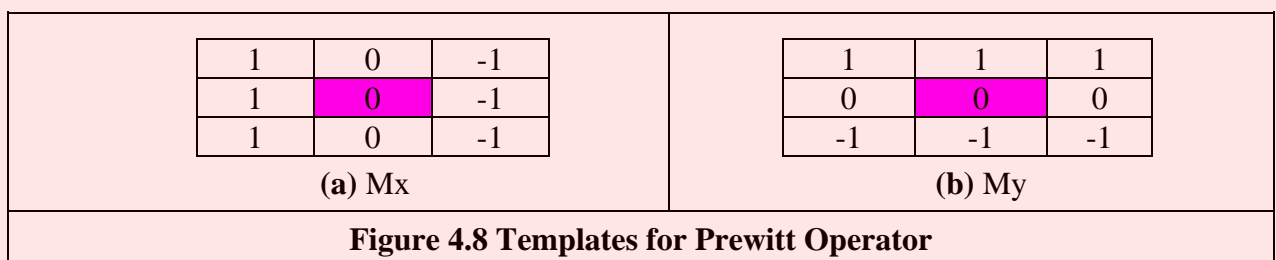
The application of the Roberts cross operator to the image of the square is shown in Figure 4.6. The two templates provide the results in Figure 4.6(a) and (b) and the result delivered by the Roberts operator is shown in Figure 4.6(c). Note that the corners of the square now appear in the edge image, by virtue of the diagonal differencing action, whereas they were less apparent in Figure 4.2(d) (where the top left corner did not appear).

An alternative to taking the maximum is to simply add the results of the two templates together to combine horizontal and vertical edges. There are of course more varieties of edges and it is often better to consider the two templates as providing components of an *edge vector*: the strength of the edge along the horizontal and vertical axes. These give components of a vector and can be added in a vectorial manner (which is perhaps more usual for the Roberts operator). The *edge magnitude* is the length of the vector, the *edge direction* is the vector's orientation, as shown in Figure 4.7.



#### 4.2.1.3 Prewitt Edge Detection Operator

Edge detection is akin to differentiation. Since it detects change it is bound to respond to noise, as well as to step-like changes in image intensity (its frequency domain analogue is high-pass filtering as illustrated in Figure 2.30(c)). It is therefore prudent to incorporate *averaging* within the edge detection process. We can then extend the vertical template,  $M_x$ , along three rows, and the horizontal template,  $M_y$ , along three columns. These give the *Prewitt edge detection operator* [Prewitt66] that consists of two templates, Figure 4.8:



This gives two results: the rate of change of brightness along each axis. As such, this is the vector illustrated in Figure 4.7: the *edge magnitude*,  $M$ , is the length of the vector and the *edge direction*,  $\theta$ , is the angle of the vector:

$$M(x, y) = \sqrt{M_x(x, y)^2 + M_y(x, y)^2} \tag{4.12}$$

$$\theta(x, y) = \tan^{-1} \left( \frac{M_y(x, y)}{M_x(x, y)} \right) \tag{4.13}$$

The signs of  $M_x$  and  $M_y$  can be used to determine the appropriate quadrant for the edge direction. An implementation of the Prewitt operator is given in Code 4.2. In this code, both templates operate on a  $3 \times 3$  region. Again, template convolution could be used to implement this operator, but (as with direct averaging and basic first order edge detection) it is less suited to small templates.

```

for x,y in itertools.product(range(0, width-1), range(0, height-1)):
    mX,mY = 0.0, 0.0
    for c in range(-1, 2):
        mX += float(inputImage[y - 1, x + c]) - float(inputImage[y + 1, x + c])
        mY += float(inputImage[y + c, x - 1]) - float(inputImage[y + c, x + 1])

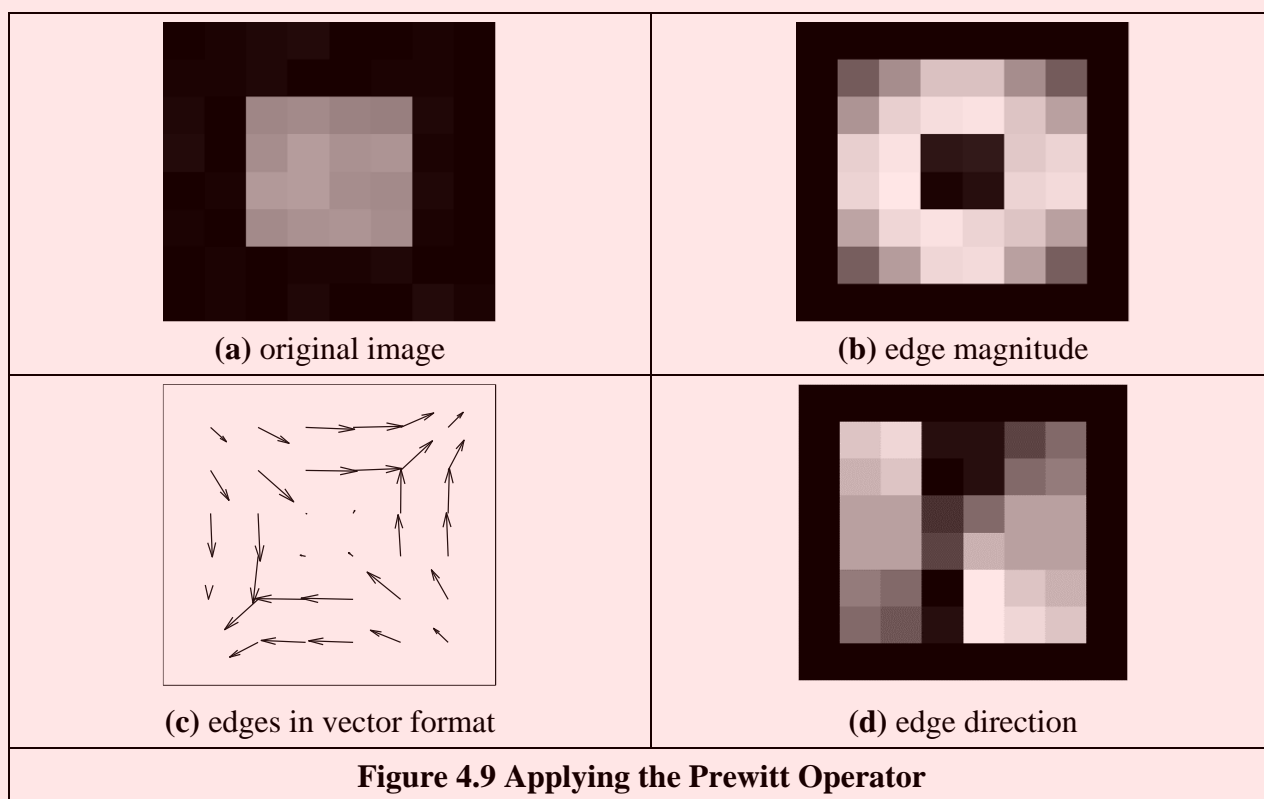
    outputMagnitude[y,x] = math.sqrt(mX * mX + mY * mY)

```

```
outputDirection[y,x] = math.atan2(mY, mX)
```

**Code 4.2 Implementing the Prewitt Operator**

When applied to the image of the square, Figure 4.9(a), we obtain the edge magnitude and direction, Figures 4.9(b) and (d), respectively. The edge direction in Figure 4.9(d) is shown as an image as the numbers are rather uninformative. Though the regions of edge points are wider due to the operator’s averaging properties, the edge data is clearer than the earlier first order operator, highlighting the regions where intensity changed in a more reliable fashion (compare for example, the lower left corner of the square which was not revealed earlier). The direction is less clear in an image format and is better exposed by Matlab’s `quiver` format in Figure 4.9(c). In vector format, the edge direction data is clearly less well defined at the corners of the square (as expected, since the first order derivative is discontinuous at these points). Again, template convolution could be used to implement this operator, but (as with direct averaging and basic first order edge detection) it is less suited to small templates.



**4.2.1.4 Sobel Edge Detection Operator**

When the weight at the central pixels, for both Prewitt templates, is doubled, this gives the famous *Sobel edge detection operator* which, again, consists of two templates to determine the edge in vector form. The Sobel operator was the most popular edge-detection operator until the development of edge detection techniques with a theoretical basis. It proved popular because it gave, overall, a better performance than other contemporaneous edge-detection operators, such as the Prewitt operator. The templates for the Sobel operator can be found in Figure 4.10.

1	0	-1		1	2	1
---	---	----	--	---	---	---

<table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">2</td> <td style="padding: 2px 10px; background-color: #ff00ff;">0</td> <td style="padding: 2px 10px;">-2</td> </tr> <tr> <td style="padding: 2px 10px;">1</td> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px;">-1</td> </tr> </table> <p>(a) <math>M_x</math></p>	2	0	-2	1	0	-1	<table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px 10px;">0</td> <td style="padding: 2px 10px; background-color: #ff00ff;">0</td> <td style="padding: 2px 10px;">0</td> </tr> <tr> <td style="padding: 2px 10px;">-1</td> <td style="padding: 2px 10px;">-2</td> <td style="padding: 2px 10px;">-1</td> </tr> </table> <p>(b) <math>M_y</math></p>	0	0	0	-1	-2	-1
2	0	-2											
1	0	-1											
0	0	0											
-1	-2	-1											
<b>Figure 4.10 Templates for Sobel Operator</b>													

The implementation of these masks is very similar to the implementation of the Prewitt operator, Code 4.2, again operating on a  $3 \times 3$  sub-picture. However, all template-based techniques can be larger than  $5 \times 5$  so, as with any group operator, there is a  $7 \times 7$  Sobel and so on. The virtue of a larger edge-detection template is that it involves more smoothing to reduce noise but edge blurring becomes a great problem. The estimate of edge direction can be improved with more smoothing since it is particularly sensitive to noise. So how do we form larger templates, say for  $5 \times 5$  or  $7 \times 7$ ? Few textbooks state the original Sobel derivation, but it has been attributed [Heath97] as originating from a PhD thesis [Sobel70]. Unfortunately a theoretical basis, that can be used to calculate the coefficients of larger templates, is rarely given. One approach to a theoretical basis is to consider the optimal forms of averaging and of differencing. Gaussian averaging has already been stated to give optimal averaging. The Binomial Expansion gives the integer coefficients of a series that, in the limit, approximates the normal distribution. Pascal's triangle gives sets of coefficients for a smoothing operator which, in the limit, approach the coefficients of a Gaussian smoothing operator. Pascal's triangle is then:

Window size						
2			1		1	
3			1	2	1	
4		1	3	3	1	
5	1	4	6	4	1	

This gives the (un-normalised) coefficients of an optimal discrete smoothing operator (it is essentially a Gaussian operator with integer coefficients). The rows give the coefficients for increasing template, or window, size. The coefficients of smoothing within the Sobel operator Figures 4.10 are those for a window size of 3. In Python, the template coefficients can be calculated as shown in Code 4.3. The code uses the array `smooth` to store a pair of 2D templates that contain the coefficients in x and y directions.

The differencing coefficients are given by Pascal's triangle for subtraction:

Window size						
2			1		-1	
3			1	0	-1	
4		1	1	-1	-1	
5	1	2	0	-2	-1	

This can be implemented by subtracting the templates derived from two adjacent Pascal expansions from a smaller window size. In Code 4.3 the arrays `pascal1` and `pascal2` have the expansions for the Pascal triangle and a shifted version. The code uses the factorial definition of the Pascal coefficients to compute both expansions, but the second one can be also computed by shifting the values of the first array. Note that each Pascal expansion has a template for vertical and horizontal direction. Figure 4.11 shows the coefficients computed with the implementation for a  $7 \times 7$  window size.

The differencing template is then given by the difference between two Pascal expansions, as shown in Code 4.3. The array `Sobel` stores the result of the difference between the two shifted templates. The result is multiplied by the smoothing array. Thus, the result gives the coefficients of optimal differencing and optimal smoothing. This *general form* of the Sobel operator combines optimal smoothing along one axis, with optimal differencing along the other.

```

for x,y in itertools.product(range(0, kernelSize), range(0, kernelSize)):

    # Smooth
    smooth[y,x,0] = factorial(kernelSize - 1) / \
                    (factorial(kernelSize - 1 - x) * factorial(x))
    smooth[y,x,1] = factorial(kernelSize - 1) / \
                    (factorial(kernelSize - 1 - y) * factorial(y))

    # Pascal
    if (kernelSize - 2 - x >= 0):
        pascal1[y,x,0] = factorial(kernelSize - 2) / \
                        (factorial(kernelSize - 2 - x) * factorial(x))

    if (kernelSize - 2 - y >= 0):
        pascal1[y,x,1] = factorial(kernelSize - 2) / \
                        (factorial(kernelSize - 2 - y) * factorial(y))

    # Pascal shift to the right
    xp = x - 1
    if (kernelSize - 2 - xp >= 0 and xp >= 0):
        pascal2[y,x,0] = factorial(kernelSize - 2) / \
                        (factorial(kernelSize - 2 - xp) * factorial(xp))

    yp = y - 1
    if (kernelSize - 2 - yp >= 0 and yp >= 0):
        pascal2[y,x,1] = factorial(kernelSize - 2) / \
                        (factorial(kernelSize - 2 - yp) * factorial(yp))

    # Sobel
    sobel[y,x,0] = smooth[y,x,1] * (pascal1[y,x,0] - pascal2[y,x,0])
    sobel[y,x,1] = smooth[y,x,0] * (pascal1[y,x,1] - pascal2[y,x,1])

```

Code 4.3 Creating Sobel Kernels

Figure 4.11 shows the full process used to generate the  $7 \times 7$  Sobel kernels. Each column shows the template computed with Code 4.3. Figure 4.11(a) shows the smoothing template. Figures 4.11(b-c) show the Pascal and shifted Pascal templates. Figure 4.11(d) shows the final kernel.

[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 1 4 5 0 -5 -4 -1 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 6 24 30 0 -30 -24 -6 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 15 60 75 0 -75 -60 -15 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 20 80 100 0 -100 -80 -20 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 15 60 75 0 -75 -60 -15 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 6 24 30 0 -30 -24 -6 ]
[ 1 6 15 20 15 6 1 ]	[ 1 5 10 10 5 1 0 ]	[ 0 1 5 10 10 5 1 ]	[ 1 4 5 0 -5 -4 -1 ]
(a) smooth $7 \times 7$	(b) Pascal $7 \times 7$	(c) shift $7 \times 7$	(d) Sobel kernel $7 \times 7$

Figure 4.11 Creating Templates for the Sobel Operator

The Sobel templates can be applied by operating on a matrix of dimension equal to the window size, from which edge magnitude and gradient are calculated. The implementation in Code 4.4 convolves the generalised Sobel templates (of size chosen to be `kernelize`) with the image supplied as argument, to give an output which contains the images of edge magnitude and direction. The kernel is created according to the implementation in Code 4.3.

```

# Create Kernel
sobelX, sobelY = createSobelKernel(kernelSize)

# The center of the kernel
kernelCentre = (kernelSize - 1) / 2

# Convolution with two kernels
for x,y in itertools.product(range(0, width), range(0, height)):

```

```

mX, wX, mY, wY = 0.0, 0.0, 0.0, 0.0
for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):
    posY = y + wy - kernelCentre
    posX = x + wx - kernelCentre

    if posY > -1 and posY < height and posX > -1 and posX < width:
        mX += float(inputImage[posY,posX]) * sobelX[wy, wx]
        wX += sobelX[wy, wx]

        mY += float(inputImage[posY,posX]) * sobelY[wy, wx]
        wY += sobelY[wy, wx]

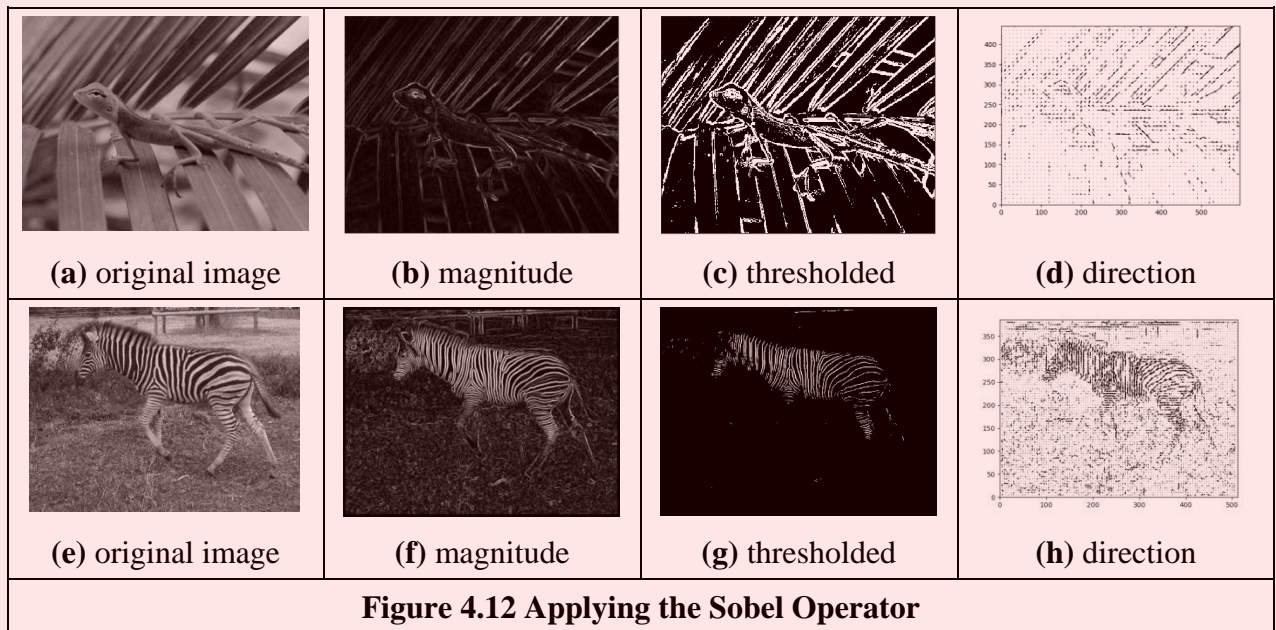
    if wX > 0: mX = mX / wX
    if wY > 0: mY = mY / wY

outputMagnitude[y,x] = math.sqrt(mX * mX + mY * mY)
outputDirection[y,x] = math.atan2(mX, -mY)

```

**Code 4.4 Generalised Sobel Operator**

The results of applying the 5×5 Sobel operator can be seen in Figure 4.12. The original image Figure 4.12(a) has many edges of the leaves and around the lizard’s eye. This is shown in the edge magnitude image, Figure 4.12(b). When this is thresholded at a suitable value, many edge points are found, as shown in Figure 4.12(c). Note that in areas of the image where the brightness remains fairly constant, such as within the blades of the leaves, there is little change which is reflected by low edge magnitude and few points in the thresholded data. The example in the second row in Figure 4.12 illustrates how edge magnitude and direction are well defined in image regions with high contrast.



**Figure 4.12 Applying the Sobel Operator**

The Sobel edge direction data can be arranged to point in different ways, as can the direction provided by the Prewitt operator. If the templates are inverted to be of the form shown in Figure 4.13, the edge direction will be inverted around both axes. If only one of the templates is inverted, the measured edge direction will be inverted about the chosen axis.

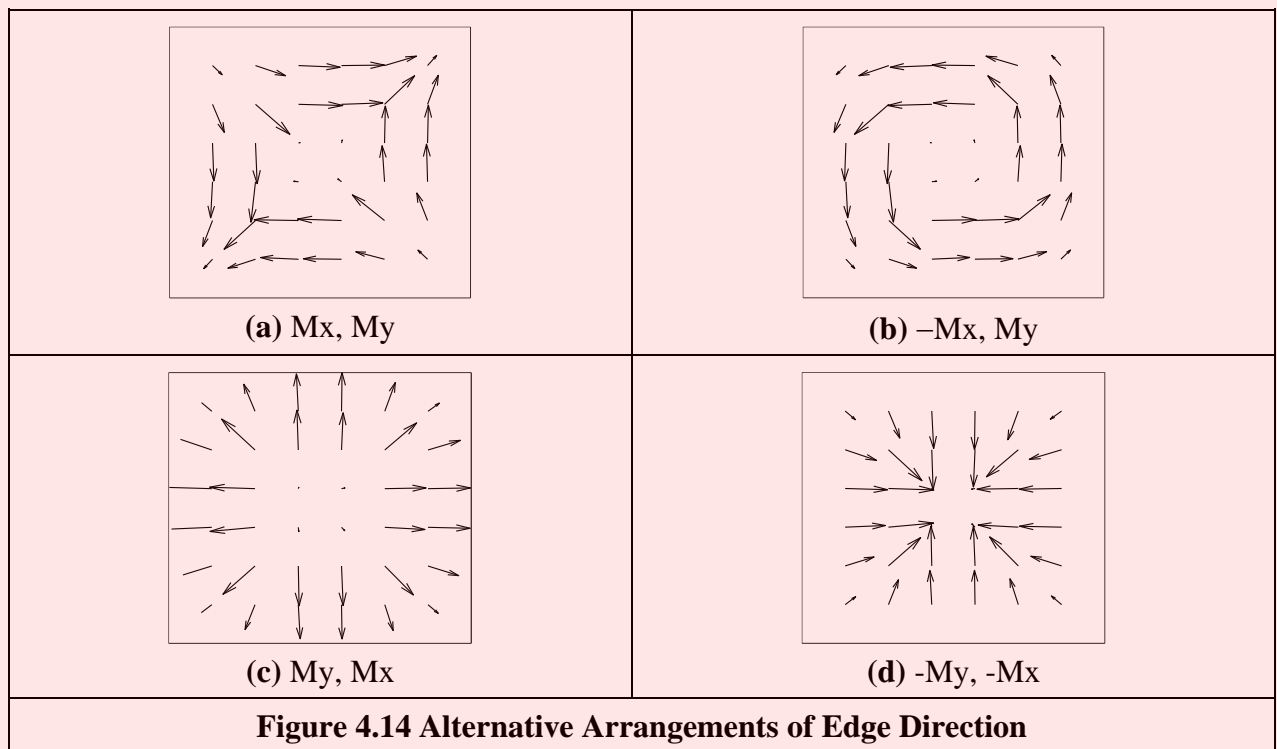
-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

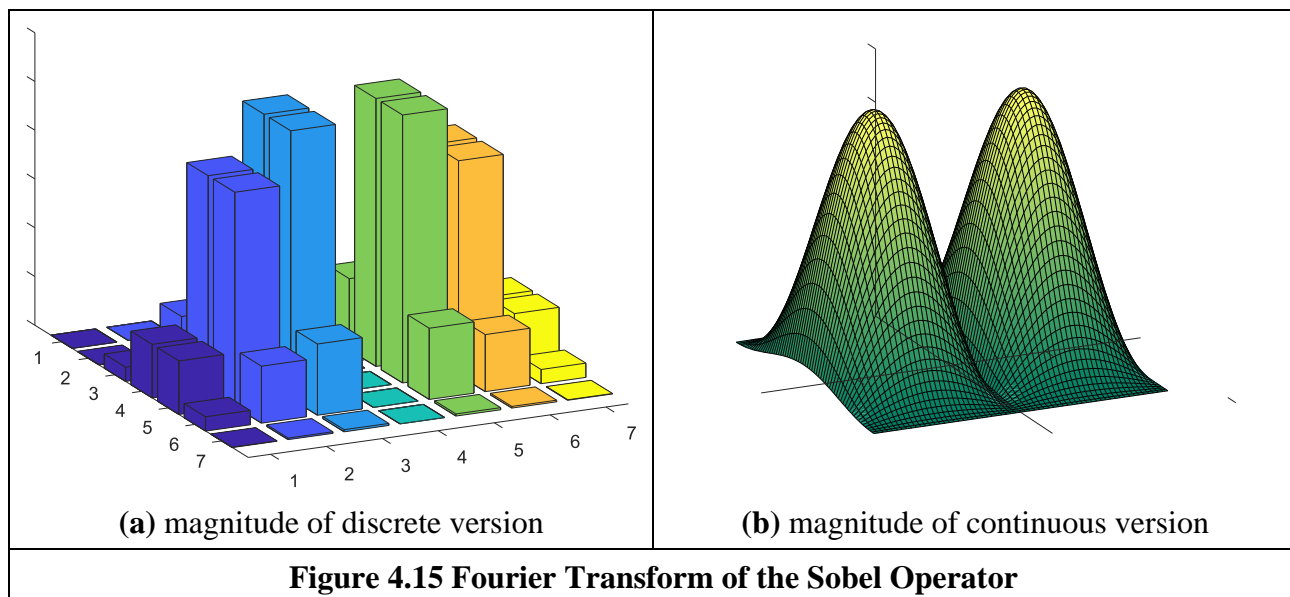


<b>(a) – Mx</b>	<b>(b) – My</b>
<b>Figure 4.13 Inverted Templates for Sobel Operator</b>	

This gives four possible directions for measurement of the *edge direction* provided by the Sobel operator, two of which (for the templates of Figures 4.10 and 4.13) are illustrated in Figures 4.14(a) and (b), respectively, where inverting the Mx template does not highlight discontinuity at the corners. (The edge magnitude of the Sobel applied to the square is not shown, but is similar to that derived by application of the Prewitt operator, Figure 4.9(b)). By swapping the Sobel templates, the measured edge direction can be arranged to be normal to the edge itself (as opposed to tangential data along the edge). This is illustrated in Figures 4.14(c) and (d) for swapped versions of the templates given in Figures 4.10 and 4.13, respectively. The rearrangement can lead to simplicity in algorithm construction when finding shapes, as to be shown later. Any algorithm which uses edge direction for finding shapes must know precisely which arrangement has been used, since the edge direction can be used to speed algorithm performance, but it must map precisely to the expected image data if used in that way.



Detecting edges by *template convolution* again has a frequency domain interpretation. The magnitude of the *Fourier transform* of a  $7 \times 7$  Sobel template of Figure 4.11 is given in Figure 4.15. The Fourier transform is given in relief in Figure 4.15(a). The template is for horizontal differencing action, Mx, In Figure 4.15(a) the vertical frequencies are selected from a region near the origin (*low-pass filtering* – consistent with averaging over five lines), whereas the horizontal frequencies are selected away from the origin (*high-pass* – consistent with the differencing action). This highlights the action of the Sobel operator: to combine smoothing along one axis with differencing along the other. In Figure 4.15(a), the smoothing is of vertical spatial frequencies whilst the differencing is of horizontal spatial frequencies.



An alternative frequency domain analysis of the Sobel can be derived via the  $z$ -transform operator. This is more the domain of signal processing courses in electronic and electrical engineering, and is included here for completeness and for linkage with signal processing. Essentially  $z^{-1}$  is a unit time-step delay operator, so  $z$  can be thought of a unit (time-step) advance, so  $f(t-\tau) = z^{-1}f(t)$  and  $f(t+\tau) = zf(t)$  where  $\tau$  is the sampling interval. Given that we have two spatial axes  $x$  and  $y$  we can then express the Sobel operator of Figure 4.13(a) using delay and advance via the  $z$ -transform notation along the two axes as

$$\begin{aligned}
 \text{Sobel}(x, y) = & \begin{matrix} -z_x^{-1}z_y^{-1} & +0 & +z_xz_y^{-1} \\ -2z_x^{-1} & +0 & +2z_x \\ -z_x^{-1}z_y & +0 & +z_xz_y \end{matrix} \quad (4.14)
 \end{aligned}$$

including zeros for the null template elements. Given that there is a standard substitution (by conformal mapping, evaluated along the frequency axis)  $z^{-1} = e^{-j\omega t}$  to transform from the time ( $z$ ) domain to the frequency domain ( $\omega$ ), then we have

$$\begin{aligned}
 \text{Sobel}(\omega_x, \omega_y) &= -e^{-j\omega_x t} e^{-j\omega_y t} + e^{j\omega_x t} e^{-j\omega_y t} - 2e^{-j\omega_x t} + 2e^{j\omega_x t} - e^{-j\omega_x t} e^{j\omega_y t} + e^{j\omega_x t} e^{j\omega_y t} \\
 &= \left( e^{-j\omega_y t} + 2 + e^{j\omega_y t} \right) \left( -e^{-j\omega_x t} + e^{j\omega_x t} \right) \\
 &= \left( e^{\frac{-j\omega_y t}{2}} + e^{\frac{j\omega_y t}{2}} \right)^2 \left( -e^{-j\omega_x t} + e^{j\omega_x t} \right) \\
 &= 8j \cos^2 \left( \frac{\omega_y t}{2} \right) \sin(\omega_x t)
 \end{aligned} \quad (4.15)$$

where the transform  $\text{Sobel}$  is a function of spatial frequency  $\omega_x$  and  $\omega_y$ , along the  $x$  and the  $y$  axes. This confirms rather nicely the separation between smoothing along the  $y$  axis (the  $\cos$  function in the first part of Equation 4.15, low-pass) and differencing along the other – here by differencing (the  $\sin$  function, high-pass) along the  $x$  axis. This provides the continuous form of the magnitude of the transform shown in Figure 4.15(b).

#### 4.2.1.5 The Canny Edge Detector

The *Canny edge detection operator* [Canny86] is perhaps the most popular edge detection technique at present. It was formulated with three main objectives:

1. optimal detection with no spurious responses;
2. good localisation with minimal distance between detected and true edge position; and
3. single response to eliminate multiple responses to a single edge.

The first requirement aims to reduce the response to noise. This can be effected by *optimal smoothing*; Canny was the first to demonstrate that Gaussian filtering is optimal for edge detection (within his criteria). The second criterion aims for accuracy: edges are to be detected, in the right place. This can be achieved by a process of *non-maximum suppression* (which is equivalent to peak detection). Non-maximum suppression retains only those points at the top of a ridge of edge data, whilst suppressing all others. This results in thinning: the output of non-maximum suppression is thin lines of edge points, in the right place. The third constraint concerns location of a single edge point in response to a change in brightness. This is because more than one edge can be denoted to be present, consistent with the output obtained by earlier edge operators.

Canny showed that the *Gaussian operator* was optimal for image smoothing. Recalling that the Gaussian operator  $g(x,y,\sigma)$  is given by:

$$g(x, y, \sigma) = e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (4.16)$$

By differentiation, for unit vectors  $U_x = [1,0]$  and  $U_y = [0,1]$  along the co-ordinate axes, we obtain:

$$\begin{aligned} \nabla g(x, y) &= \frac{\partial g(x, y, \sigma)}{\partial x} U_x + \frac{\partial g(x, y, \sigma)}{\partial y} U_y \\ &= -\frac{x}{\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} U_x - \frac{y}{\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} U_y \end{aligned} \quad (4.17)$$

Equation 4.17 gives a way to calculate the coefficients of a *derivative of Gaussian* template that combines first-order differentiation with Gaussian smoothing. This is a smoothed image, and so the edge will be a ridge of data. In order to mark an edge at the correct point (and to reduce multiple responses), we can convolve an image with an operator which gives the first derivative in a direction normal to the edge. The maximum of this function should be the peak of the edge data, where the gradient in the original image is sharpest, and hence the location of the edge. Accordingly, we seek an operator,  $G_n$ , which is a first derivative of a Gaussian function  $g$  in the direction of the normal,  $\mathbf{n}_\perp$ :

$$G_n = \frac{\partial g}{\partial \mathbf{n}_\perp} \quad (4.18)$$

where  $\mathbf{n}_\perp$  can be estimated from the first order derivative of the Gaussian function  $g$  convolved with the image  $\mathbf{P}$ , and scaled appropriately as:

$$\mathbf{n}_\perp = \frac{\nabla(\mathbf{P} * g)}{|\nabla(\mathbf{P} * g)|} \quad (4.19)$$

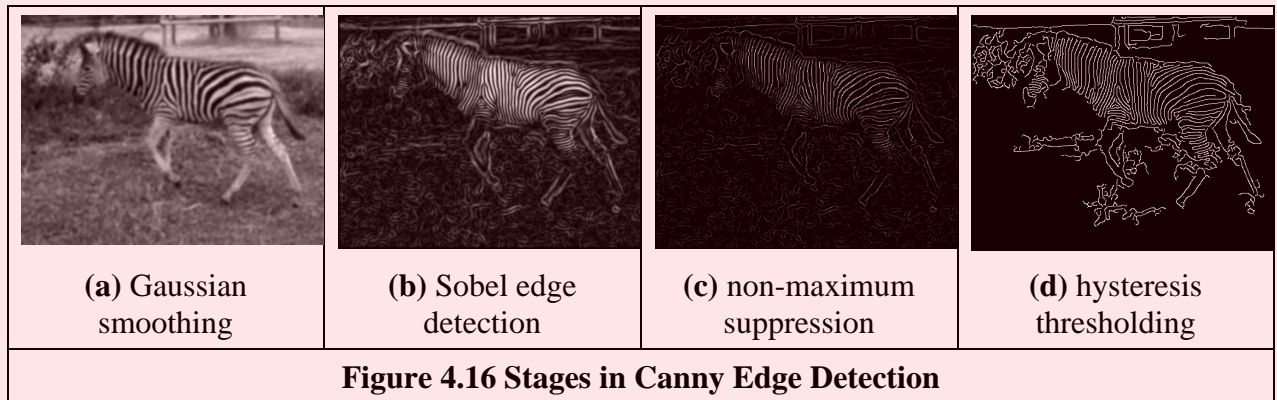
The location of the true edge point is then at the maximum point of  $G_n$  convolved with the image. This maximum is when the differential (along  $\mathbf{n}_\perp$ ) is zero:

$$\frac{\partial(G_n * \mathbf{P})}{\partial \mathbf{n}_\perp} = 0 \quad (4.20)$$

By substitution of Equation 4.18 in Equation 4.20,

$$\frac{\partial^2(G * \mathbf{P})}{\partial \mathbf{n}_\perp^2} = 0 \quad (4.21)$$

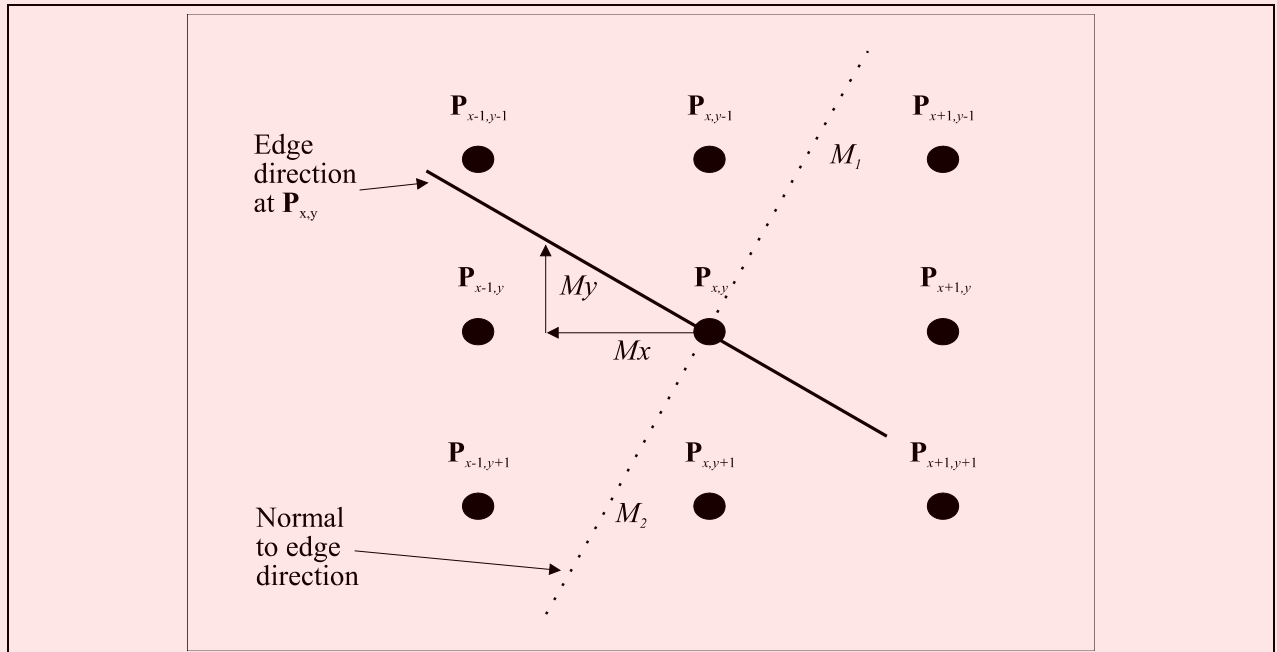
Equation 4.21 provides the basis for an operator which meets one of Canny's criteria, namely that edges should be detected in the correct place. This is non-maximum suppression, which is equivalent to retaining peaks (and thus equivalent to differentiation perpendicular to the edge), which thins the response of the edge-detection operator to give edge points which are in the right place, without multiple response and with minimal response to noise. However, it is virtually impossible to achieve an exact implementation of Canny given the requirement to estimate the normal direction.



A common approximation is, as illustrated in Figure 4.16:

1. use *Gaussian smoothing* (as in Section 3.4.5), Figure 4.16(a);
2. use the *Sobel operator*, Figure 4.16(b);
3. use *non-maximum suppression*, Figure 4.16(c); and
4. threshold with *hysteresis* to connect edge points, Figure 4.16(d).

Note that the first two stages can be combined using a version of Equation 4.17, but are separated here so that all stages in the edge-detection process can be shown clearly. An alternative implementation of Canny's approach [Deriche87] used Canny's criteria to develop two-dimensional recursive filters, claiming performance and implementation advantage over the approximation here.


**Figure 4.17 Interpolation in Non-Maximum Suppression**

*Non-maximum suppression* essentially locates the highest points in the edge magnitude data. This is performed by using edge direction information, to check that points are at the peak of a ridge. Given a  $3 \times 3$  region, a point is at a maximum if the gradient at either side of it is less than the gradient at the point. This implies that we need values of gradient along a line which is normal to the edge at a point. This is illustrated in Figure 4.17, which shows the neighbouring points to the point of interest,  $\mathbf{P}_{x,y}$ , the edge direction at  $\mathbf{P}_{x,y}$  and the normal to the edge direction at  $\mathbf{P}_{x,y}$ . The point  $\mathbf{P}_{x,y}$  is to be marked as maximum if its gradient,  $M(x,y)$  exceeds the gradient at points 1 and 2,  $M_1$  and  $M_2$ , respectively. Since we have a discrete neighbourhood,  $M_1$  and  $M_2$  need to be interpolated, First order interpolation using  $M_x$  and  $M_y$  at  $\mathbf{P}_{x,y}$ , and the values of  $M_x$  and  $M_y$  for the neighbours gives:

$$M_1 = \frac{M_y}{M_x} M(x+1, y-1) + \frac{M_x - M_y}{M_x} M(x, y-1) \quad (4.22)$$

and

$$M_2 = \frac{M_y}{M_x} M(x-1, y+1) + \frac{M_x - M_y}{M_x} M(x, y+1) \quad (4.23)$$

The point  $\mathbf{P}_{x,y}$  is then marked as a maximum if  $M(x,y)$  exceeds both  $M_1$  and  $M_2$ , otherwise it is set to zero. In this manner the peaks of the ridges of edge magnitude data are retained, whilst those not at the peak are set to zero. Code 4.5 illustrates the implementation of non-maximum suppression. It takes as inputs the images `magnitude` and `angle` that were obtained by applying the Sobel edge detector. The code iterates for all pixel positions  $(x, y)$  whose magnitude is bigger than a minimum threshold. Any point lower than this threshold cannot be an edge. The variable `normalAngle` stores the angle perpendicular to the edge. This angle is used to obtain the two pixels defined by the variables  $(x1, y1)$  and  $(x2, y2)$ . These points are used to compute  $M_1$  and  $M_2$ . If the magnitude in the point  $(x, y)$  is greater than these values, then we consider that the point is a maximum. The resulting `maxImage` will be zero if the point is not maximum and the value of the magnitude otherwise.

```
border = (kernelSize - 1) / 2
```

```

for x,y in itertools.product(range(border, width-border),
                             range(border, height-border)):

    # Only potential edges can be maximum
    if magnitude[y,x] > lowerT:
        # The normal angle is perpendicular to the edge angle
        normalAngle = angle[y,x] - pi / 2.0

        # Make sure the angle is between 0 and pi
        while normalAngle < 0:    normalAngle += pi
        while normalAngle > pi:  normalAngle -= pi

        # Angle defining the first point
        baseAngle = int( 4 * normalAngle / pi ) * (pi / 4.0)

        # Integer delta positions for interpolation
        # We use -y since the image origin is in top corner
        x1, y1 = int(round(cos(baseAngle))), -int(round(sin(baseAngle)))
        x2, y2 = int(round(cos(baseAngle + pi / 4.0))),
                 -int(round(sin(baseAngle + pi / 4.0)))

        # How far we are from (x1,y1).
        # Maximum difference is pi / 4.0, so we multiply by 2
        w = cos(2.0*(normalAngle - baseAngle))

        # Point to interpolate
        M1 = w * magnitude[y+y1,x+x1] + (1.0 - w) * magnitude[y+y2,x+x2]

        # Point to interpolate for pixels in the other side of the edge
        M2 = w * magnitude[y-y1,x-x1] + (1.0 - w) * magnitude[y-y2,x-x2]

        # Determine if it is a maximum. If so make sure it will be preserved
        if magnitude[y,x] > M1 and magnitude[y,x] > M2:
            maxImage[y,x] = magnitude[y,x]

```

#### Code 4.5 Non-Maximum Suppression

The result of non-maximum suppression is used in the hysteresis thresholding process. As illustrated in Figure 4.18, this process delineates the borders by choosing the maximum pixels. The transfer function associated with *hysteresis thresholding* is shown in Figure 4.18. Points are set to white (edges) once the upper threshold is exceeded and set to black when the lower threshold is reached. The arrows reflect possible movement: there is only one way to change from black to white and vice versa.

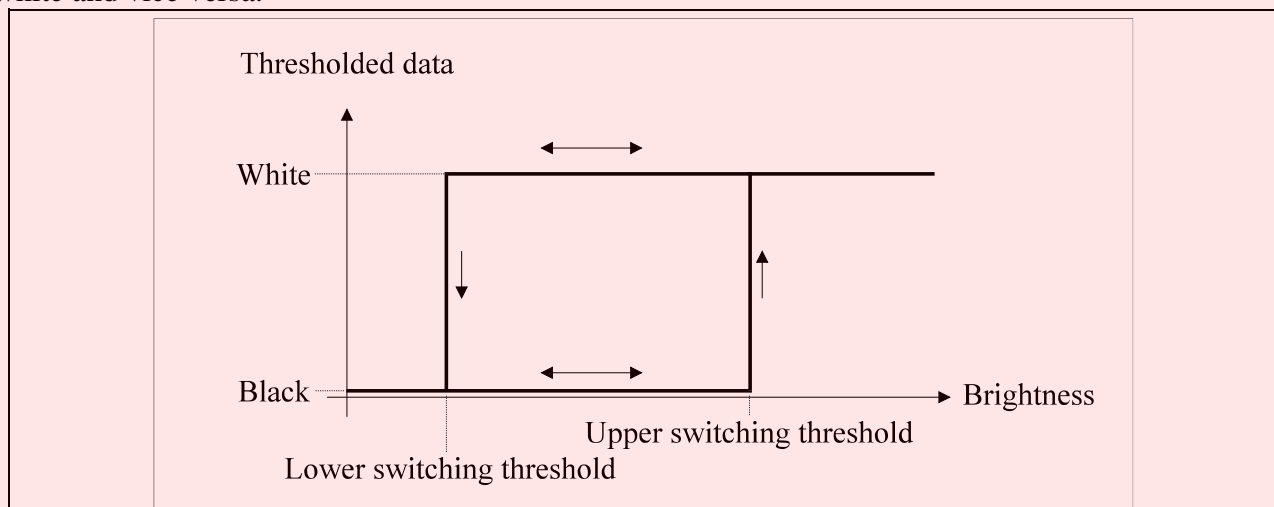
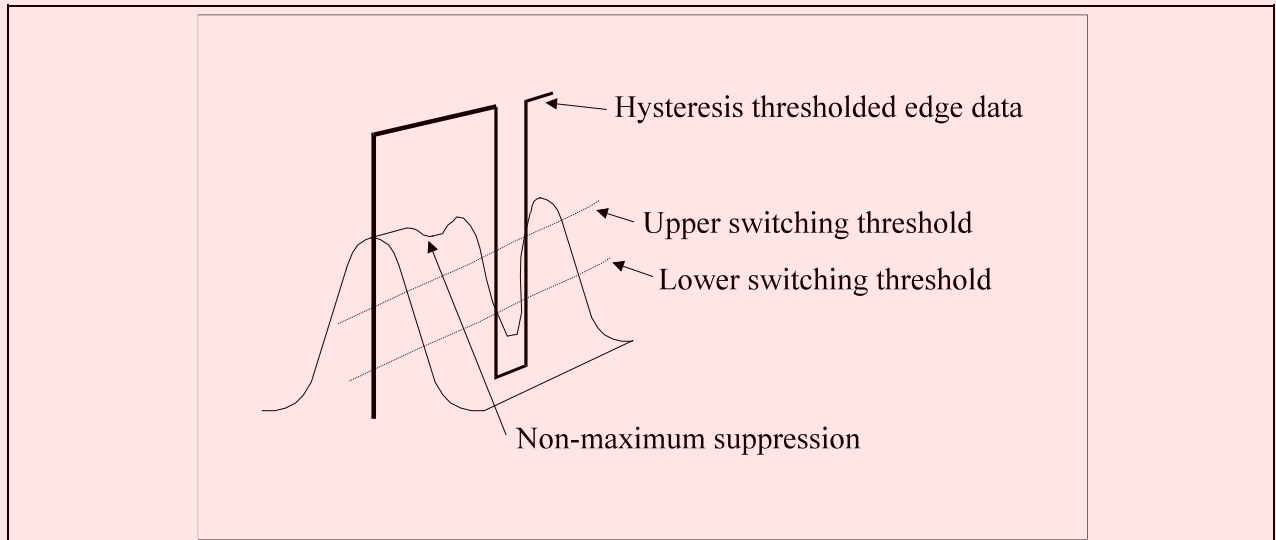


Figure 4.18 Hysteresis Thresholding Transfer Function

The application of non-maximum suppression and hysteresis thresholding is illustrated in Figure 4.19. This contains a ridge of edge data, the edge magnitude. The action of non-maximum suppression is to select the points along the top of the ridge. Given that the top of the ridge initially exceeds the upper threshold, the thresholded output is set to white until the peak of the ridge falls beneath the lower threshold. The thresholded output is then set to black until the peak of the ridge exceeds the upper switching threshold.



**Figure 4.19 Action of Non-Maximum Suppression and Hysteresis Thresholding**

Hysteresis thresholding requires two thresholds: an upper and a lower threshold. The process starts when an edge point from non-maximum suppression is found to exceed the upper threshold. This is labelled as an edge point (usually white, with a value 255) and forms the first point of a line of edge points. The neighbours of the point are then searched to determine whether or not they exceed the lower threshold, as in Figure 4.20. Any neighbour that exceeds the lower threshold is labelled as an edge point and its neighbours are then searched to determine whether or not they exceed the lower threshold. In this manner, the first edge point found (the one that exceeded the upper threshold) becomes a seed point for a search. Its neighbours, in turn, become seed points if they exceed the lower threshold, and so the search extends, along branches arising from neighbours that exceeded the lower threshold. For each branch, the search terminates at points that have no neighbours above the lower threshold.

$\geq$ lower	$\geq$ lower	$\geq$ lower
$\geq$ lower	seed $\geq$ upper	$\geq$ lower
$\geq$ lower	$\geq$ lower	$\geq$ lower

**Figure 4.20 Neighbourhood Search for Hysteresis Thresholding**

The implementation of *hysteresis thresholding* clearly requires recursion to delineate the edges by repeatedly selecting the neighbours of pixels that have been labelled as edges. Having found the

initial seed point, it is set to white and its neighbours are searched. Code 4.6 illustrates an implementation of this process. First, the pixels in the edge image are labelled, according to the values of the two thresholds, as edges (with a value of 255), no edges (with a value of 0) or as points that may be edges (with a value of 128). The code visits all pixels to find initial seeds. For each seed found, the code looks for the neighbours that exceed the lower threshold. These neighbours are labelled as edges and stored in the list `potentialEdges`. Recursion is implemented by looking for edges in the neighbours of each element in the list. That is, the code uses the list to keep the edges grown from the seed.

```
# Divide pixels as edges, no edges and unassigned
for x,y in itertools.product(range(1, width-1), range(1, height-1)):
    # These are edges
    if maxImage[y,x] > upperT: edges[y,x] = 255
    # These are pixels that we do not want as edges
    if maxImage[y,x] < lowerT: edges[y,x] = 0
    # These may be edges
    if maxImage[y,x] > lowerT and maxImage[y,x] <= upperT:
        edges[y,x] = 128

# Show double threshold image
showImageF(edges)

# Resolve the potential edges
for x,y in itertools.product(range(1, width-1), range(1, height-1)):
    # For each edge
    if edges[y,x] == 255:

        # Examine neighbors
        potentialEdges = [ ]
        for wx,wy in itertools.product(range(-windowDelta, windowDelta+1), \
                                       range(-windowDelta, windowDelta+1)):

            # It becomes an edge
            if edges[y+wy,x+wx] == 128:
                edges[y+wy,x+wx] = 255
                potentialEdges.append((y+wy,x+wx))

        # Look into new edges
        while len(potentialEdges) > 0:
            # Take element from potential edges
            y, x = (potentialEdges[0])[0], (potentialEdges[0])[1]
            potentialEdges = potentialEdges[1:]

            # Examine neighbor
            for wx,wy in itertools.product(range(-windowDelta, windowDelta+1), \
                                           range(-windowDelta, windowDelta+1)):

                # It becomes an edge
                if edges[y+wy,x+wx] == 128:
                    edges[y+wy,x+wx] = 255
                    potentialEdges.append((y+wy,x+wx))

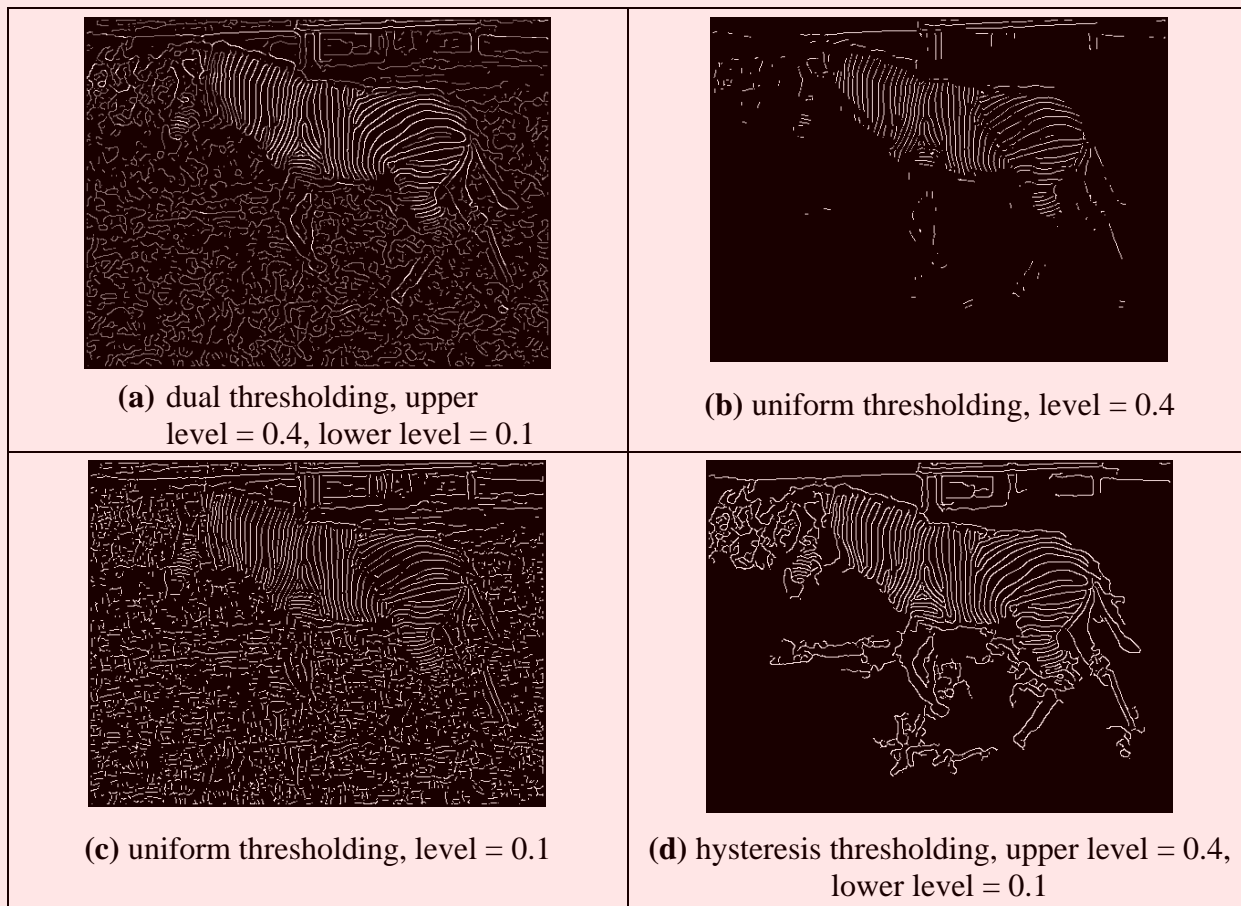
# Clean up remaining potential edges
for x,y in itertools.product(range(1, width-1), range(1, height-1)):
    if edges[y,x] == 128: edges[y,x] = 0
```

#### Code 4.6 Hysteresis Thresholding Process

A comparison between *uniform thresholding* and hysteresis thresholding is shown in Figure 4.21. Figure 4.21(a) shows the edges for dual thresholding of a Sobel edge detected image. This image shows edges with an upper threshold set to 0.4 and a lower threshold of 0.1 of the normalised edge magnitude. Edges larger than the lower threshold are shown in grey and edges larger than the upper threshold in white. This example shows that it is difficult to choose a single threshold that characterises adequately the edges in the image. Figure 4.21(b) and (c) show the result of uniform

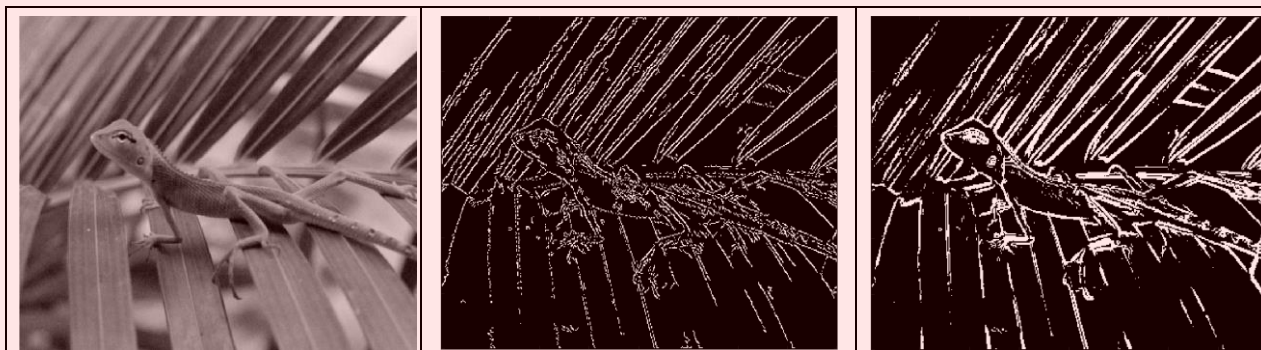


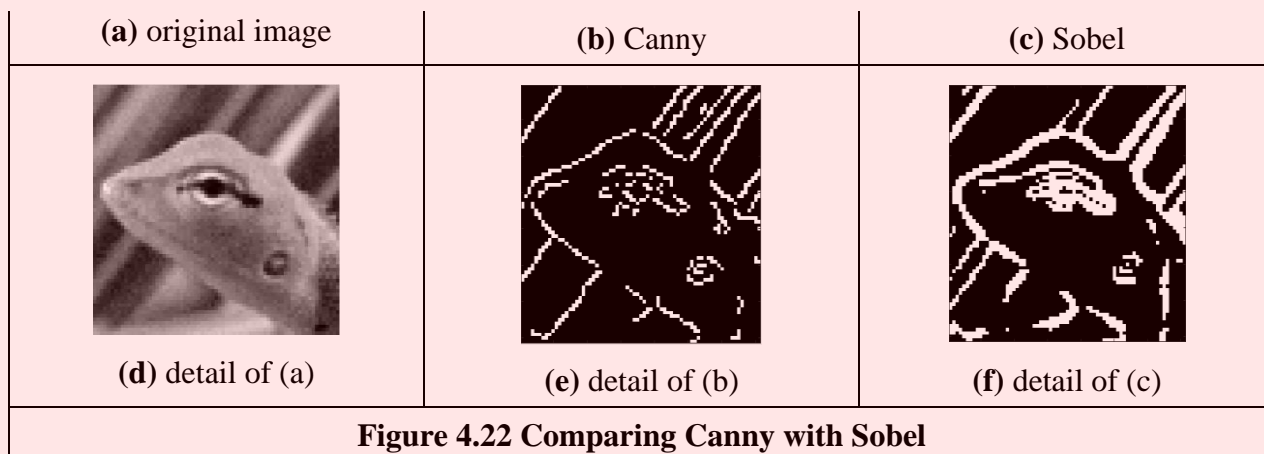
thresholding with these two thresholds. Notice that uniform thresholding can select too few points if the threshold is too high or too many if it is too low. Figure 4.21(d) shows that Hysteresis thresholding naturally selects all the points in Figure 4.21(b) and some of those in Figure 4.21(c), those connected to the points in Figure 4.21(b). Hysteresis thresholding therefore has an ability to detect major features of interest in the edge image, in an improved manner to uniform thresholding.



**Figure 4.21 Comparing Hysteresis Thresholding with Uniform Thresholding**

The action of the Canny operator is shown in Figure 4.22, in comparison with the result of the Sobel operator. Figure 4.22(a) is the original image of a face, Figure 4.22(b) is the result of the Canny operator (using a  $5 \times 5$  Gaussian operator with  $\sigma = 1.0$  and with upper and lower thresholds set appropriately) and Figure 4.22(c) is the result of a  $3 \times 3$  Sobel operator with uniform thresholding. The retention of major detail by the Canny operator is very clear; the edges are thin and complete in Figure 4.22(b) whereas they are blurred and less complete in Figure 4.22(c). Closer inspection of the lizard's eye shows greater detail too.

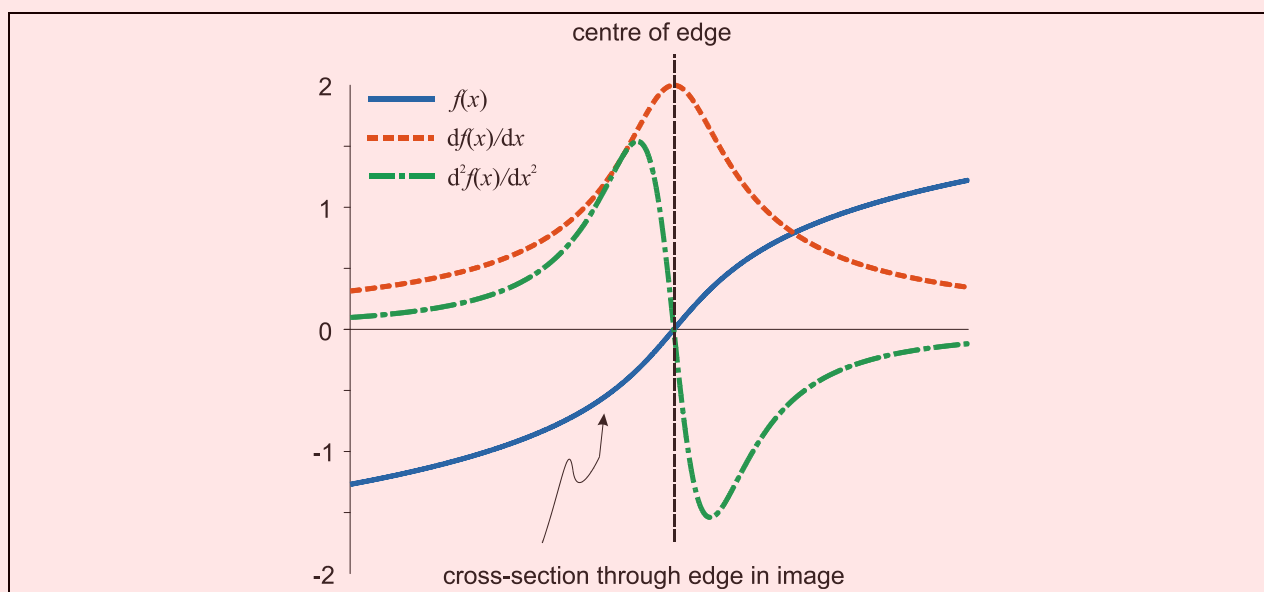




### 4.2.2 Second Order Edge Detection Operators

#### 4.2.2.1 Motivation

First order edge detection is based on the premise that differentiation highlights change; image intensity changes in the region of a feature boundary. The process is illustrated in Figure 4.23 where there is a cross-section through image data which is an edge. The result of first order edge detection,  $f'(x) = df/dx$ , is a peak where the rate of change of the original signal,  $f(x)$ , is greatest. There are of course higher order derivatives; applied to the same cross-section of data, the second order derivative,  $f''(x) = d^2f/dx^2$ , is greatest where the rate of change of the signal is greatest and zero when the rate of change is constant. The rate of change is constant at the peak of the first order derivative. This is where there is a zero-crossing in the second order derivative, where it changes sign. Accordingly, an alternative to first order differentiation is to apply second order differentiation and then find zero-crossings in the second order information.



**Figure 4.23 First and Second Order Edge Detection**

### 4.2.2.2 Basic Operators: The Laplacian

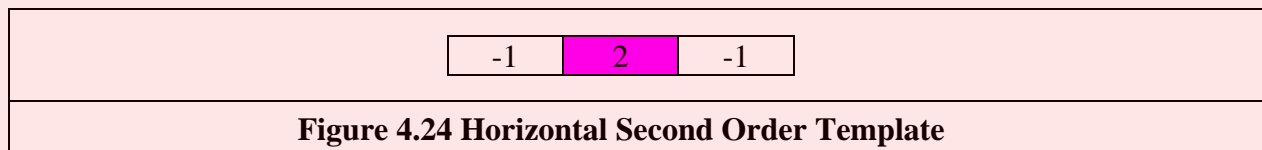
The *Laplacian operator* is a template which implements second order differencing. The second order differential can be approximated by the difference between two adjacent first order differences:

$$f''(x) \cong f'(x) - f'(x+1) \tag{4.24}$$

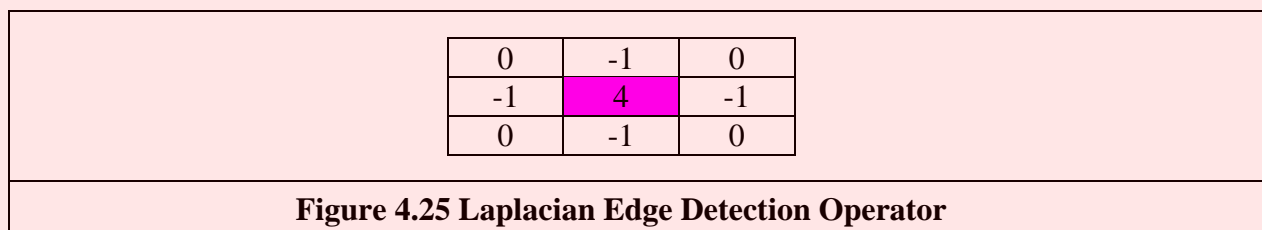
Which, by Equation 4.6, gives

$$f''(x+1) \cong -f(x) + 2f(x+1) - f(x+2) \tag{4.25}$$

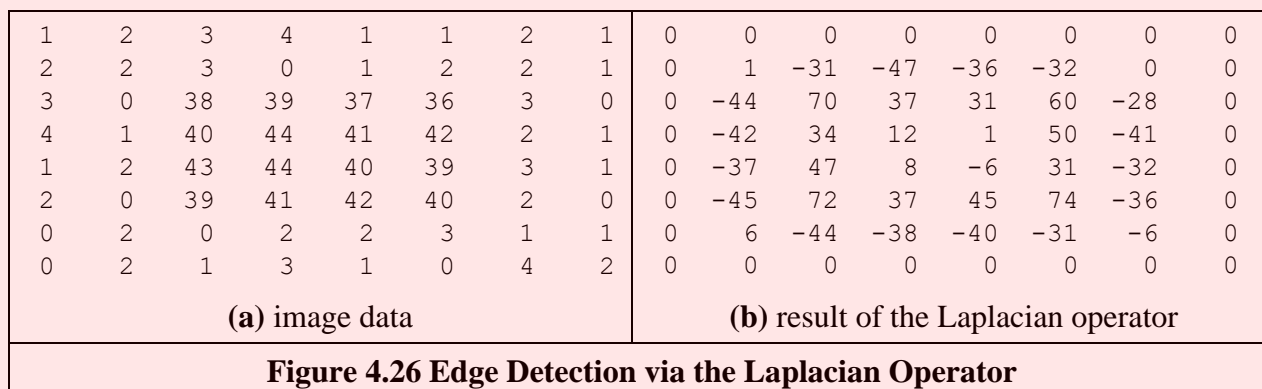
This gives a horizontal second-order template as given in Figure 4.24.



When the horizontal second order operator is combined with a vertical second order difference we obtain the full Laplacian template, given in Figure 4.25. Essentially, this computes the difference between a point and the average of its four direct neighbours. This was the operator used earlier in anisotropic diffusion, Section 3.5.5, where it is an approximate solution to the heat equation.



Application of the Laplacian operator to the image of the square is given in Figure 4.26. The original image is provided in numeric form in Figure 4.26(a). The detected edges are the zero crossings in Figure 4.26(b) and can be seen to lie between the edge of the square and its background. The result highlights the boundary of the square in the original image, but there is also a slight problem: there is a small hole in the shape in the lower right. This is by virtue of second order differentiation, which is inherently more susceptible to noise. Accordingly, to handle noise we need to introduce smoothing.



An alternative structure to the template in Figure 4.25 is one where the central weighting is 8 and the neighbours are all weighted as -1. Naturally, this includes a different form of image information, so the effects are slightly different. (Essentially, this now computes the difference between a pixel and the average of its neighbouring points, including the corners.) In both structures, the central weighting can be negative and that of the four or the eight neighbours can be positive, without loss

of generality. Actually, it is important to ensure that the sum of template coefficients is zero, so that edges are not detected in areas of uniform brightness. One advantage of the Laplacian operator is that it is isotropic (like the Gaussian operator): it has the same properties in each direction. However, as yet it contains no smoothing and will again respond to noise, more so than a first-order operator since it is differentiation of a higher order. As such, the Laplacian operator is rarely used in its basic form. Smoothing can use the averaging operator described earlier but a more optimal form is Gaussian smoothing. When this is incorporated within the Laplacian we obtain a *Laplacian of Gaussian (LoG) operator* which is the basis of the *Marr-Hildreth* approach, to be considered next. A clear disadvantage with the Laplacian operator is that edge direction is not available. It does however impose low computational cost, which is its main advantage. Though interest in the Laplacian operator abated with rising interest in the Marr-Hildreth approach, a nonlinear Laplacian operator was developed [Vliet89] and shown to have good performance, especially in low-noise situations.

#### 4.2.2.3 The Marr-Hildreth Operator

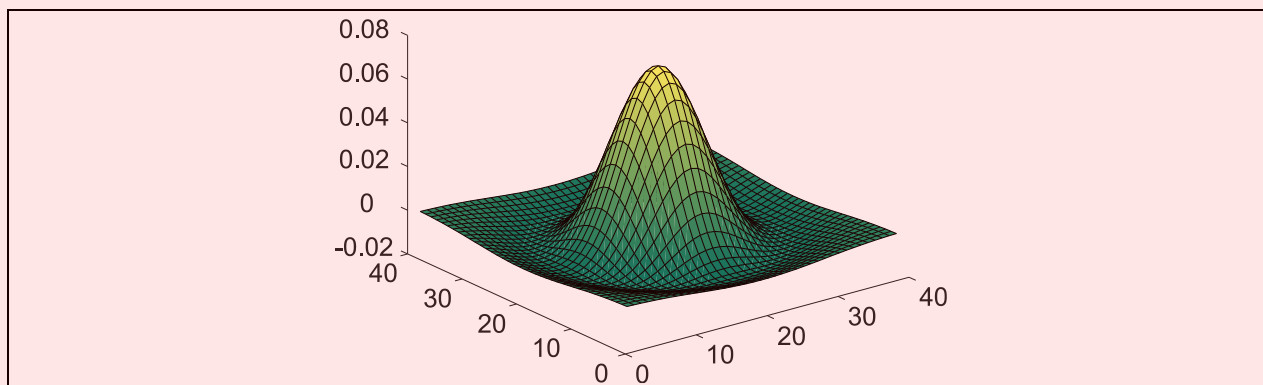
The *Marr-Hildreth* approach [Marr80] again uses Gaussian filtering. In principle, we require an image which is the second differential  $\nabla^2$  of a Gaussian operator  $g(x,y)$  convolved with an image  $\mathbf{P}$ . This convolution process can be separated as:

$$\nabla^2 (g(x, y) * \mathbf{P}) = \nabla^2 (g(x, y)) * \mathbf{P} \quad (4.26)$$

Accordingly, we need to compute a template for  $\nabla^2 (g(x, y))$  and convolve this with the image. By further differentiation of Equation 4.17, we achieve a *Laplacian of Gaussian (LoG) operator* :

$$\begin{aligned} \nabla^2 g(x, y) &= \frac{\partial^2 g(x, y, \sigma)}{\partial x^2} U_x + \frac{\partial^2 g(x, y, \sigma)}{\partial y^2} U_y \\ &= \frac{\partial \mathcal{N} g(x, y, \sigma)}{\partial x} U_x + \frac{\partial \mathcal{N} g(x, y, \sigma)}{\partial y} U_y \\ &= \left( \frac{x^2}{\sigma^2} - 1 \right) \frac{e^{-\frac{(x^2+y^2)}{2\sigma^2}}}{\sigma^2} + \left( \frac{y^2}{\sigma^2} - 1 \right) \frac{e^{-\frac{(x^2+y^2)}{2\sigma^2}}}{\sigma^2} \\ &= \frac{1}{\sigma^2} \left( \frac{(x^2 + y^2)}{\sigma^2} - 2 \right) e^{-\frac{(x^2+y^2)}{2\sigma^2}} \end{aligned} \quad (4.27)$$

This is the basis of the Marr-Hildreth operator. Equation 4.27 can be used to calculate the coefficients of a template which, when convolved with an image, combines *Gaussian smoothing* with second order differentiation. The operator is sometimes called a ‘Mexican hat’ operator, since its surface plot is the shape of a sombrero, as illustrated in Figure 4.27.

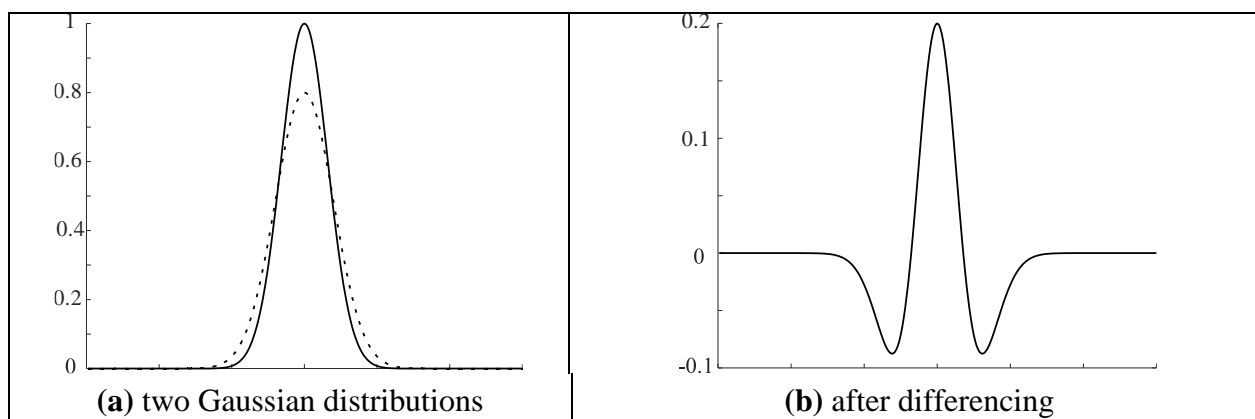


**Figure 4.27 Shape of Laplacian of Gaussian Operator**

The calculation of the Laplacian of Gaussian can be approximated by the difference of Gaussians where the difference is formed from two Gaussian filters with differing variance [Marr82, Lindeberg94].

$$\sigma \nabla^2 g(x, y, \sigma) = \frac{\partial g}{\partial \sigma} = \lim_{k \rightarrow 1} \frac{g(x, y, k\sigma) - g(x, y, \sigma)}{k\sigma - \sigma} \quad (4.28)$$

where  $g(x, y, \sigma)$  is the Gaussian function and  $k$  is a constant. Although similarly named, the derivative of Gaussian, Equation 4.17, is a first order operator including Gaussian smoothing,  $\nabla g(x, y)$ . It does actually seem counter-intuitive that the difference of two smoothing operators should lead to second order edge detection. The approximation is illustrated in Figure 4.28 where in 1-D two Gaussian distributions of different variance are subtracted to form a 1-D operator whose cross-section is similar to the shape of the LoG operator (a cross-section of Figure 4.27).



**Figure 4.28 Approximating the Laplacian of Gaussian by Difference of Gaussian**

The implementation of Equation 4.27 to calculate the template coefficients for the LoG\_template operator is given in Code 4.7. The function includes a normalisation function which ensures that the sum of the template coefficients is unity, so that edges are not detected in area of uniform brightness. This is in contrast with the earlier Laplacian operator (where the template coefficients summed to zero) since the LoG operator includes smoothing within the differencing action, whereas the Laplacian is pure differencing. The template generated by this function can then be used within template convolution. The Gaussian operator again suppresses the influence of points away from the centre of the template, basing differentiation on those points nearer the centre; the standard deviation,  $\sigma$ , is chosen to ensure this action. Again, it is isotropic consistent with Gaussian smoothing.

```

function snd_order = LoG_template(winsize,sigma)

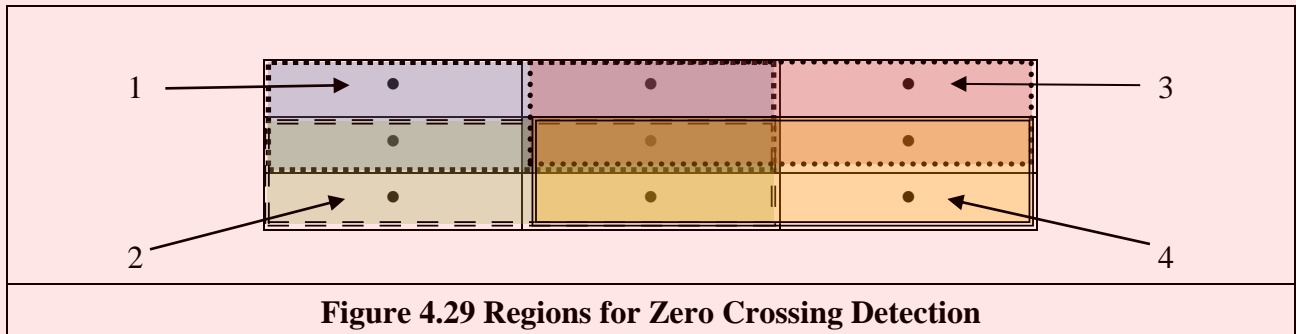
half=floor(winsize/2)+1;
snd_order(1:winsize,1:winsize)=0;

%then form the LoG template
sum=0;
for x = 1:winsize %address all columns
    for y = 1:winsize %address all rows except border, Eq. 4.27
        snd_order(y,x)= (1/(sigma^2))*(((x-half)^2+(y-half)^2/sigma^2)-2)*...
            exp(-((x-half)^2+(y-half)^2)/(2*sigma^2));
        sum=sum+snd_order(y,x);
    end
end
snd_order=snd_order/sum;
end

```

**Code 4.7 Implementation of the Laplacian of Gaussian Operator**

Determining the *zero-crossing* points is a major difficulty with this approach. There is a variety of techniques which can be used, including manual determination of zero-crossings or a least squares fit of a plane to local image data, which is followed by determination of the point at which the plane crosses zero, if it does. The former is too simplistic, whereas the latter is quite complex.



**Figure 4.29 Regions for Zero Crossing Detection**

The approach here is much simpler: given a local  $3 \times 3$  area of an image, this is split into quadrants. These are shown in Figure 4.29 where each quadrant contains the centre pixel. The first quadrant contains the four points in the upper left corner and the third quadrant contains the four points in the upper right. If the average of the points in any quadrant differs in sign from the average in any other quadrant, there must be a zero crossing at the centre point. In `zero_cross`, Code 4.8, the total intensity in each quadrant is evaluated, giving four values `int(1)` to `int(4)`. If the maximum value of these points is positive, and the minimum value is negative, there must be a zero-crossing within the neighbourhood. If one exists, the output image at that point is marked as white, otherwise it is set to black.

The action of the Marr-Hildreth operator is given in Figure 4.30, applied to the lizard image in Figure 4.22(a). The output of convolving the LoG operator is hard to interpret visually and is not shown here (remember that it is the zero-crossings which mark the edge points and it is hard to see them). The detected zero-crossings (for a  $3 \times 3$  neighbourhood) are shown in Figures 4.30(b) and (c) for LoG operators of size  $9 \times 9$  with  $\sigma = 1.4$  and  $19 \times 19$  with  $\sigma = 3.0$ , respectively. These show that the selection of window size and variance can be used to provide edges at differing scales. Some of the smaller regions in Figure 4.30(b) join to form larger regions in Figure 4.30(c). Given the size of these templates, it is usual to use a Fourier implementation of template convolution (Section 3.4.4). Note that one virtue of the Marr-Hildreth operator is its ability to provide closed edge borders which the Canny operator cannot. Another virtue is that it avoids the recursion associated with hysteresis thresholding that can require a massive stack size for large images.

```

function zero_xing = zero_cross(image)
%New image shows zero crossing points

%get dimensions
[rows,cols]=size(image);

%set the output image to black (0)
zero_xing(1:rows,1:cols)=0;

%then form the four quadrant points
for x = 2:cols-1 %address all columns except border
  for y = 2:rows-1 %address all rows except border
    int(1)=image(y-1,x-1)+image(y-1,x)+image(y,x-1)+image(y,x);
    int(2)=image(y,x-1)+image(y,x)+image(y+1,x-1)+image(y+1,x);
    int(3)=image(y-1,x)+image(y-1,x+1)+image(y,x)+image(y,x+1);
    int(4)=image(y,x)+image(y,x+1)+image(y+1,x)+image(y+1,x+1);
    %now work out the max and min values
    maxval=max(int);
    minval=min(int);
    %and label it as zero crossing if there is one (!)
    if (maxval>0)&&(minval<0)
      zero_xing(y,x)=255;
    end
  end
end
end

```

Code 4.8 Zero Crossing Detector

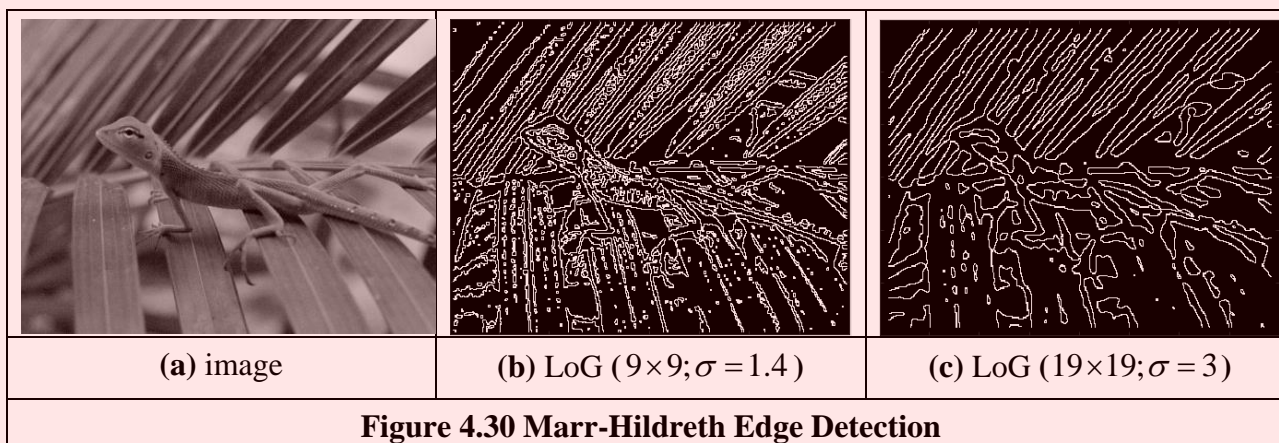
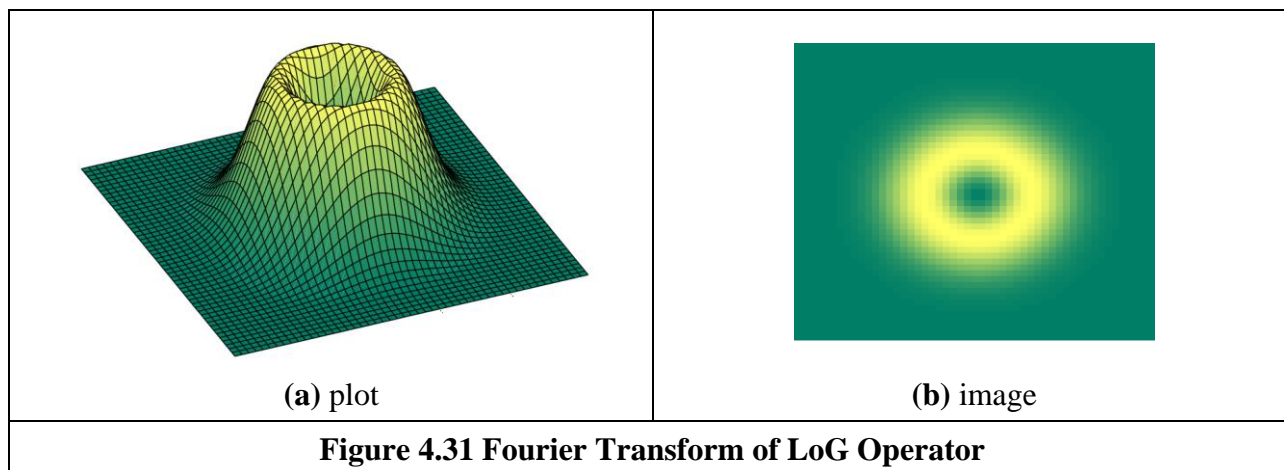


Figure 4.30 Marr-Hildreth Edge Detection

The Fourier Transform of a (large) LoG operator is shown in relief in Figure 4.31(a) and as an image in Figure 4.31(b). The transform is circular-symmetric, as expected. Since the transform reveals that the LoG operator omits low and high frequencies (those close to the origin, and those far away from the origin) it is equivalent to a *band-pass filter*. Choice of the value of  $\sigma$  controls the spread of the operator in the spatial domain and the ‘width’ of the band in the frequency domain: setting  $\sigma$  to a high value gives *low-pass* filtering, as expected. This differs from first-order edge detection templates which offer a high-pass (differencing) filter along one axis with a low-pass (smoothing) action along the other axis.



The Marr-Hildreth operator has stimulated much attention, perhaps in part because it has an appealing relationship to human vision, and its ability for multiresolution analysis (the ability to detect edges at differing scales). In fact, it has been suggested that the original image can be reconstructed from the zero-crossings at different scales. One early study [Haralick84] concluded that the Marr-Hildreth operator could give good performance. Unfortunately, the implementation appeared to be different from the original LoG operator (and has actually appeared in some texts in this form) as noted by one of the Marr-Hildreth study's originators [Grimson85]. This led to a somewhat spirited reply [Haralick85] clarifying concern but also raising issues about the nature and operation of edge detection schemes which remain relevant today. Given the requirement for convolution of large templates, attention quickly focused on frequency domain implementation [Huertas86], and speed improvement was later considered in some detail [Forshaw88]. Later, schemes were developed to refine the edges produced via the LoG approach [Ulupinar90]. Though speed and accuracy are major concerns with the Marr-Hildreth approach, it is also possible for zero-crossing detectors to mark as edge points ones which have no significant contrast, motivating study of their authentication [Clark89]. Gunn studied the relationship between mask size of the LoG operator and its error rate [Gunn99]. Essentially, an acceptable error rate defines a truncation error which in turn gives an appropriate mask size. Gunn also observed the paucity of studies on zero-crossing detection and offered a detector slightly more sophisticated than the one here (as it includes the case where a zero-crossing occurs at a boundary whereas the one here assumes that the zero-crossing can only occur at the centre). The similarity is not co-incidental: Mark developed the one here after conversations with Steve Gunn, who he works with!

### 4.2.3 Other Edge Detection Operators

There have been many approaches to edge detection. This is not surprising since it is often the first stage in a vision process. The most popular operators are the Sobel, Canny and Marr-Hildreth operators. Clearly, in any implementation there is a compromise between (computational) cost and efficiency. In some cases, it is difficult to justify the extra complexity associated with the Canny and the Marr-Hildreth operators. This is in part due to the images: few images contain the adverse noisy situations that complex edge operators are designed to handle. Also, when finding shapes, it is often prudent to extract more than enough low-level information, and to let the more sophisticated shape detection process use, or discard, the information as appropriate. For these reasons we will study only three more edge detection approaches, and only briefly. Two of these operators are the *Spacek* and the *Petrou* operators: both are designed to be optimal and both have different properties and a different basis (the smoothing functional in particular) to the Canny and Marr-Hildreth approaches. The Spacek and Petrou operators are included by virtue of their optimality. Essentially,



whilst Canny maximised the ratio of the signal to noise ratio with the localisation, Spacek [Spacek86] maximised the ratio of the product of the signal to noise ratio and the peak separation with the localisation. In Spacek's work, since the edge was again modelled as a step function, the ideal filter appeared to be of the same form as Canny's. Spacek's operator can give better performance than Canny's formulation [Jia95], as such challenging the optimality of the Gaussian operator for noise smoothing (in step edge detection), though such advantage should be explored in application.

Petrou questioned the validity of the step-edge model for real images [Petrou91]. Given that the composite performance of an image acquisition system can be considered to be that of a low-pass filter, any step changes in the image will be smoothed to become a ramp. As such, a more plausible model of the edge is a ramp rather than a step. Since the process is based on ramp edges, and because of limits imposed by its formulation, the Petrou operator uses templates that are much wider in order to preserve optimal properties. As such, the operator can impose greater computational complexity but is a natural candidate for applications with the conditions for which its properties were formulated.

Of the other approaches, Korn developed a unifying operator for symbolic representation of grey level change [Korn88]. The *Susan* operator [Smith97] derives from an approach aimed to find more than just edges since it can also be used to derive *corners* (where feature boundaries change direction sharply, as in *curvature* detection in Section 4.4.1) and structure-preserving image noise reduction. Essentially, SUSAN derives from Smallest Univalued Segment Assimilating Nucleus which concerns aggregating the difference between elements in a (circular) template centred on the nucleus. The USAN is essentially the number of pixels within the circular mask which have similar brightness to the nucleus. The edge strength is then derived by subtracting the USAN size from a geometric threshold, which is say  $\frac{3}{4}$  of the maximum USAN size. The method includes a way of calculating edge direction, which is essential if non-maximum suppression is to be applied. The advantages are in simplicity (and hence speed), since it is based on simple operations, and the possibility of extension to find other feature types.

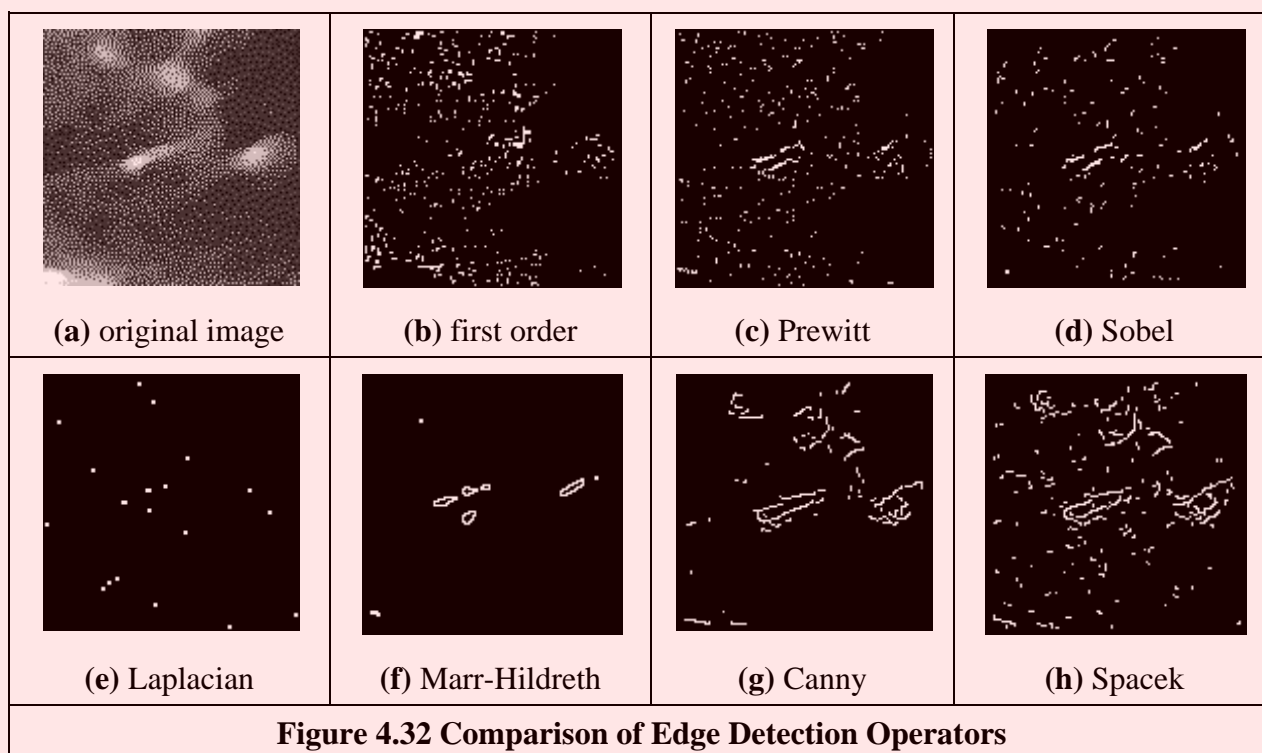


Figure 4.32 Comparison of Edge Detection Operators

#### 4.2.4 Comparison of Edge Detection Operators

Naturally, the selection of an edge operator for a particular application depends on the application itself. As has been suggested, it is not usual to require the sophistication of the advanced operators in many applications. This is reflected in analysis of the performance of the edge operators on the images here. In order to provide a different basis for comparison, we shall consider the difficulty of low-level feature extraction in ultrasound images. As has been seen earlier (Section 3.5), ultrasound images are very noisy and require filtering prior to analysis. Figure 4.32(a) is part of the ultrasound image which could have been filtered using the truncated median operator (Section 3.5.2). The image contains a feature called the pitus (it's the "splodge" in the middle) and we shall see how different edge operators can be used to detect its perimeter, though without noise filtering. The median is a very popular filtering processes for general (i.e. non-ultrasound) applications. Accordingly, it is of interest that one study [Bovik87] has suggested that the known advantages of median filtering (the removal of noise with the preservation of edges, especially for salt and pepper noise) are shown to good effect if it is used as a prefilter to first- and second- order approaches, though naturally with the cost of the median filter. However, we will not consider median filtering here: its choice depends more on suitability to a particular application.

The results for all edge operators have been generated using hysteresis thresholding where the thresholds were selected manually for best performance. The basic first-order operator, Figure 4.32(b), responds rather nicely to the noise and it is difficult to select a threshold which reveals a major part of the pitus border. Some is present in the Prewitt and Sobel operators' results, Figure 4.32(c) and Figure 4.32(d) respectively, but there is still much noise in the processed image, though there is less in the Sobel – as expected. The Laplacian operator, Figure 4.32(e), gives very little information indeed, as to be expected with such noisy imagery. However, the more advanced operators can be used to good effect. The Marr-Hildreth approach improves matters, Figure 4.32(f), but suggests that it is difficult to choose a LoG operator of appropriate size to detect a feature of these dimensions in such noisy imagery – illustrating the compromise between the size of operator needed for noise filtering and the size needed for the target feature. However, the Canny and Spacek operators can be used to good effect, as shown in Figures 4.32(g) and (h) respectively. These reveal much of the required information, together with data away from the pitus itself. In an automated analysis system, for this application, the extra complexity of the more sophisticated operators would clearly be warranted.

#### 4.2.5 Further Reading on Edge Detection

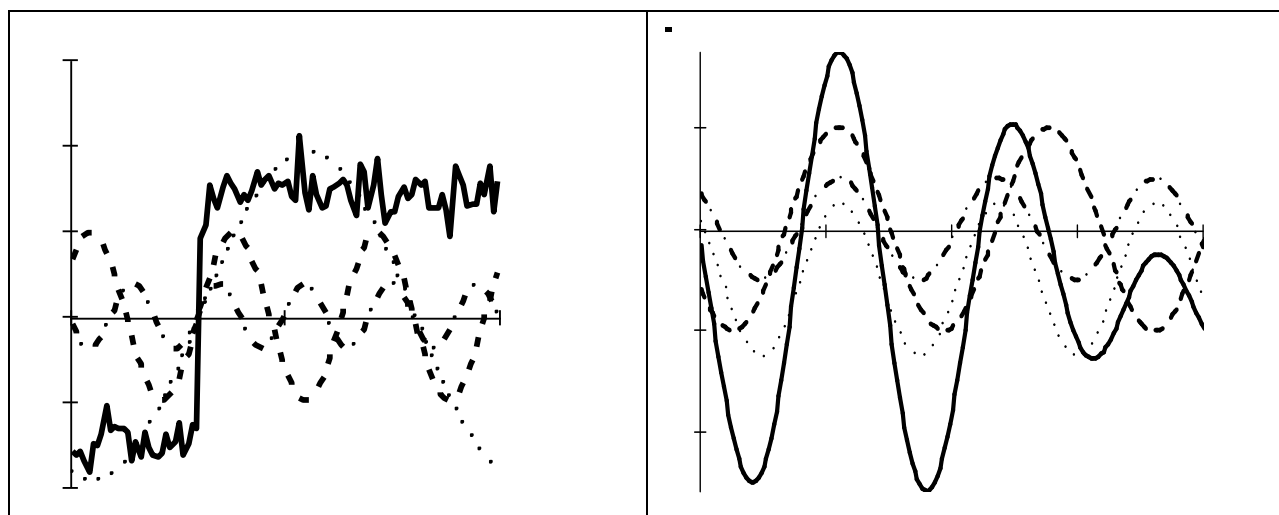
Few computer vision and image processing texts omit detail concerning edge-detection operators, though few give explicit details concerning implementation. Naturally, many of the earlier texts omit the bases of techniques. Further information can be found in journal papers; Petrou's excellent study of edge detection [Petrou94] highlights study of the performance factors involved in the optimality of the Canny, Spacek and Petrou operators with extensive tutorial support (though Petrou junior might have been embarrassed by the frequency his mugshot was used as his teeth showed up very well!). There have been a number of surveys of edge detection highlighting performance attributes in comparison. See for example [Torre86] that gives a theoretical study of edge detection and considers some popular edge detection techniques in light of this analysis. One survey, [Heath97] reviews many approaches, comparing them in particular with the Canny operator (and states where code for some of the techniques they compared can be found). This showed that best results can be achieved by tuning an edge detector for a particular application and highlighted good results by the Bergholm operator [Bergholm87]. Marr, [Marr82], considers the Marr-Hildreth

approach to edge detection in the light of human vision (and its influence on perception), with particular reference to scale in edge detection. [Yitzhaky03] suggests “a general tool to assist in practical implementations of parametric edge detectors where an automatic process is required” and uses statistical tests to evaluate edge detector performance. Since edge detection is one of the most important vision techniques, it continues to be a focus of research interest. In this regard: one recent paper used a patch based approach aimed at accuracy and computational efficiency [Dollár15]; another has been aimed to improve edge direction accuracy [Kimia18]. More recently, as we shall find in Chapter 12, deep learning approaches can base information flow on oriented edge detectors through an analysis network [LeCun15]. Accordingly, it is always worth looking at recent papers to find new techniques, or perhaps more likely performance comparison or improvements, that might help you solve a problem.

### 4.3 Phase Congruency

The comparison of edge detectors highlights some of their innate problems: incomplete contours; the need for selective thresholding; and their response to noise. Further, the selection of a threshold is often inadequate for all the regions in an image since there are many changes in local illumination. We shall find that some of these problems can be handled at a higher level, when shape extraction can be arranged to accommodate partial data and to reject spurious information. There is though natural interest in refining the low-level feature extraction techniques further.

*Phase congruency* is a feature detector with two main advantages: it can detect a broad range of features; and it is invariant to local (and smooth) change in illumination. As the name suggests, it is derived by frequency domain considerations operating on the considerations of phase (a.k.a. time). It is illustrated detecting some one-dimensional features in Figure 4.33 where the features are the solid lines: a (noisy) step function in Figure 4.33(a), and a peak (or impulse) in Figure 4.33(b). By Fourier transform analysis, any function is made up from the controlled addition of sine waves of differing frequencies. For the step function to occur (the solid line in Figure 4.33(a)), the constituent frequencies (the dotted lines in Figure 4.33(a)) must all change at the same time, so they add up to give the edge. Similarly, for the peak to occur, then the constituent frequencies must all peak at the same time; in Figure 4.33(b) the solid line is the peak and the dotted lines are some of its constituent frequencies. This means that in order to find the feature we are interested in, we can determine points where events happen at the same time: this is phase congruency. By way of generalisation, a triangle wave is made of peaks and troughs: phase congruency implies that the peaks and troughs of the constituent signals should coincide.



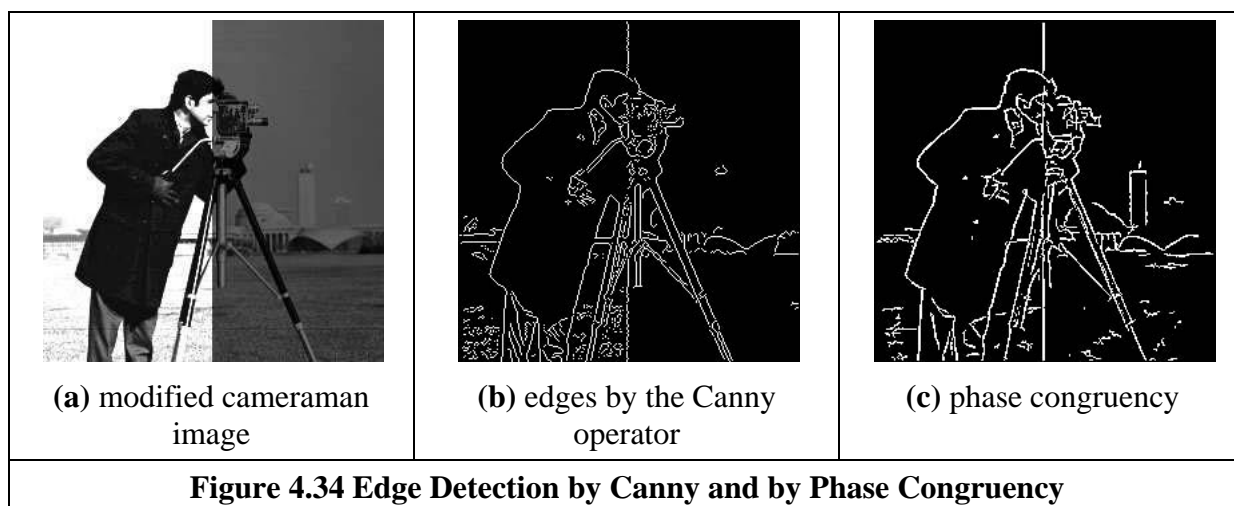
(a) step edge	(b) peak
<b>Figure 4.33 Low-level Feature Extraction by Phase Congruency</b>	

In fact, the constituent sine waves plotted in Figure 4.33(a) were derived by taking the Fourier transform and then determining the sine waves according to their magnitude and phase. The Fourier transform in Equation 2.15 delivers the complex Fourier components  $\mathbf{Fp}$ . These can be used to show the constituent signals  $xc$  by

$$xc(t) = |\mathbf{Fp}_u| e^{j\left(\frac{2\pi}{N}ut + \phi(\mathbf{Fp}_u)\right)} \tag{4.29}$$

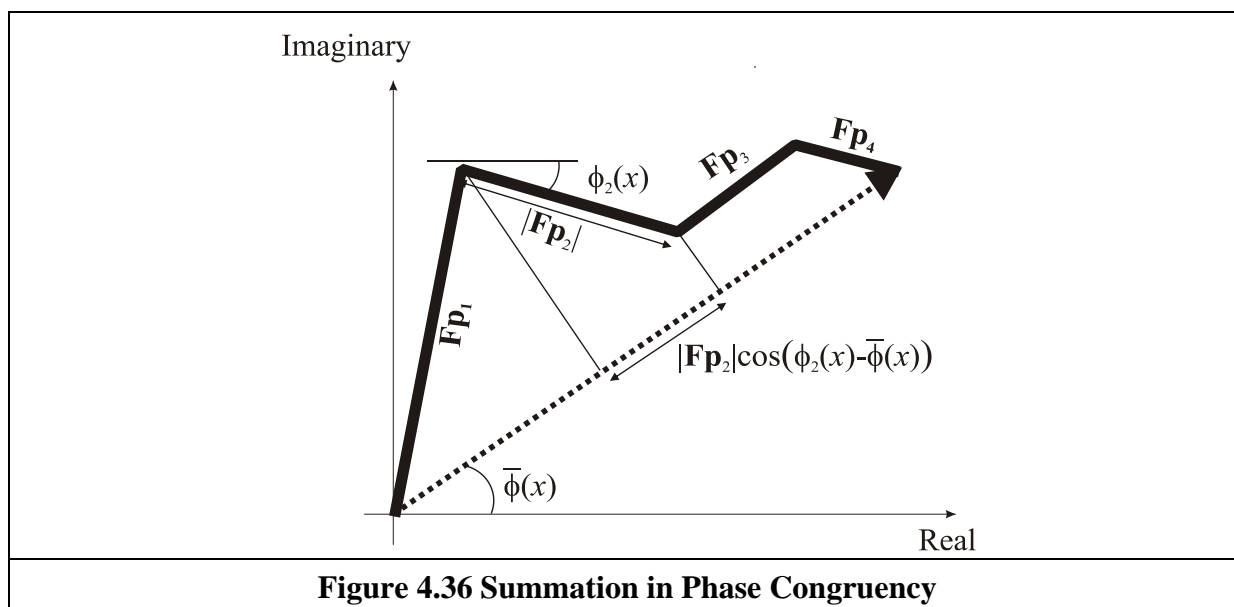
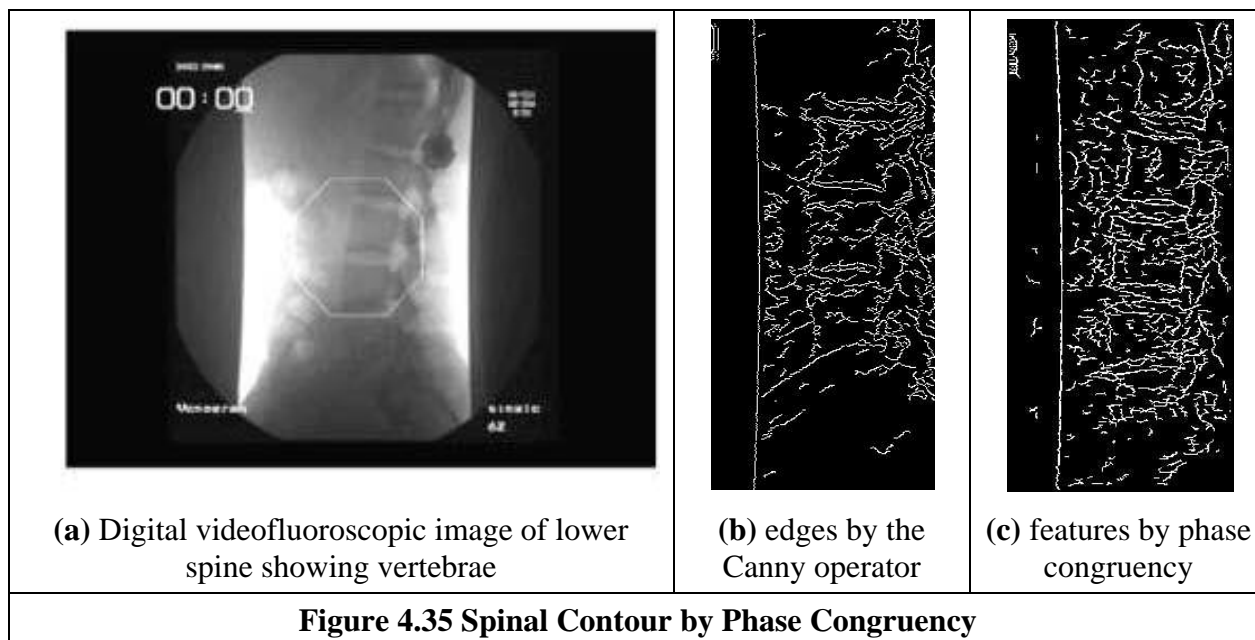
where  $|\mathbf{Fp}_u|$  is again the magnitude of the  $u^{\text{th}}$  Fourier component (Equation 2.7) and  $\phi(\mathbf{Fp}_u) = \langle \mathbf{Fp}_u$  is the argument, the phase in Equation 2.8. The (dotted) frequencies displayed in Figure 4.33 are the first four odd components (the even components for this function are zero, as shown in the Fourier transform of the step in Figure 2.11). The addition of these components is indeed the *inverse Fourier transform* which reconstructs the step feature.

The advantages are that detection of congruency is invariant with local contrast: the sine waves still add up so the changes are still in the same place, even if the magnitude of the step edge is much smaller. In images, this implies that we can change the contrast and still detect edges. This is illustrated in Figure 4.34. Here a standard image processing image, the “cameraman” image from the early UCSD dataset has been changed between the left and right side so the contrast changes in the two halves of the image, Figure 4.34(a). Edges detected by *Canny* are shown in Figure 4.34(b) and by phase congruency in 4.34(c). The basic structure of the edges detected by phase congruency is very similar to that structure detected by Canny, and the phase congruency edges appear somewhat cleaner (there is a single line associated with the tripod control in phase congruency); both detect the change in brightness between the two halves. There is a major difference though: the building in the lower right side of the image is barely detected in the Canny image whereas it can clearly be seen by phase congruency. Its absence is due to the parameter settings used in the Canny operator. These can be changed, but if the contrast were to change again, then the parameters would need to be re-optimised for the new arrangement. This is not the case for phase congruency.



Naturally such a change in brightness might appear unlikely in practical applications, but this is not the case with moving objects which interact with illumination or in fixed applications where illumination changes. In studies aimed to extract spinal information from digital videofluoroscopic X-ray images in order to provide guidance for surgeons [Zheng04], phase congruency was found

to be immune to the changes in contrast caused by slippage of the shield used to protect the patient while acquiring the image information. One such image is shown in Figure 4.35. The lack of shielding is apparent in the bloom at the side of the images. This changes as the subject is moved so it proved difficult to optimise the parameters for Canny over the whole sequence, Figure 4.35(b), but the detail of a section of the phase congruency result, Figure 4.35(c), shows that the vertebrae information is readily available for later high-level feature extraction.

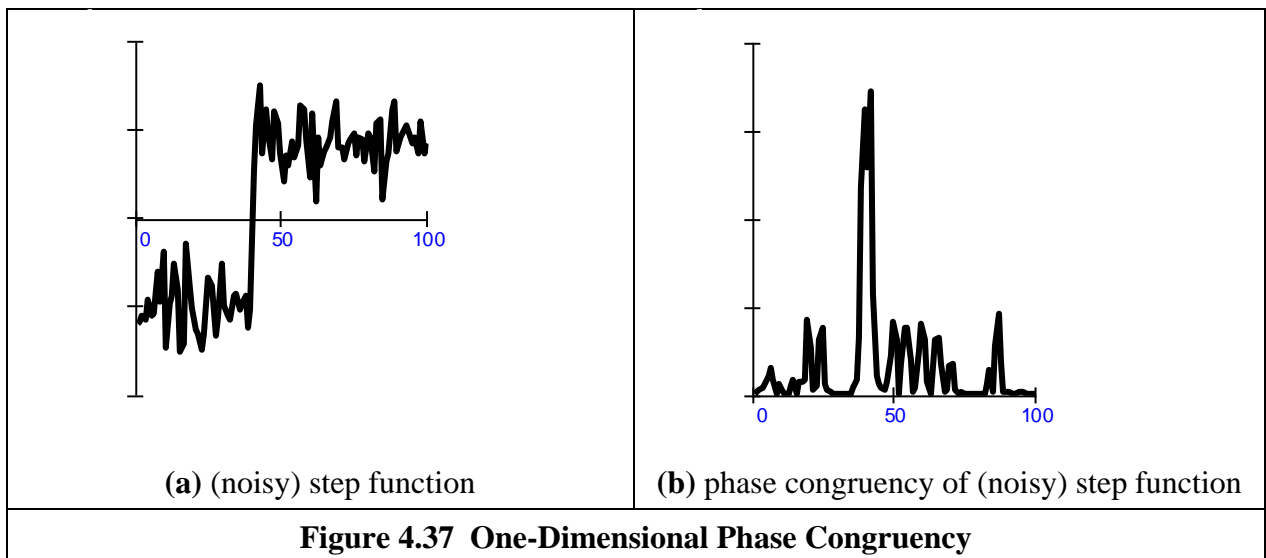


The original notions of phase congruency are the concepts of *local energy* [Morrone87], with links to the human visual system [Morrone88]. One of the most sophisticated implementations was by Kovese [Kovese99], with added advantage that his Matlab implementation is available on the Web (<https://www.peterkovese.com/matlabfns/>) as well as much more information. Essentially we seek to determine features by detection of points at which Fourier components are maximally in phase. By extension of the Fourier reconstruction functions in Equation 4.29, Morrone and Owens defined a measure of phase congruency *PC* as

$$PC(x) = \max_{\bar{\phi}(x) \in [0, 2\pi)} \left( \frac{\sum_u |\mathbf{Fp}_u| \cos(\phi_u(x) - \bar{\phi}(x))}{\sum_u |\mathbf{Fp}_u|} \right) \quad (4.30)$$

where  $\phi_u(x)$  represents the local phase of the component  $\mathbf{Fp}_u$  at position  $x$ . Essentially this computes the ratio of the sum of projections onto a vector (the sum in the numerator) to the total vector length (the sum in the denominator). The value of  $\bar{\phi}(x)$  that maximizes this equation is the amplitude weighted mean local phase angle of all the Fourier terms at the point being considered. In Figure 4.36 the resulting vector is made up of four components, highlighting the projection of the second onto the resulting vector. Clearly, the value of  $PC$  ranges from 0 to 1, the maximum occurring when all elements point along the resulting vector. As such, the resulting phase congruency is a dimensionless normalized measure which is thresholded for image analysis.

In this way, we have calculated the phase congruency for the step function in Figure 4.37(a) which is shown in Figure 4.37(b). Here, the position of the step is at time step 40; this is the position of the peak in phase congruency, as required. Note that the noise can be seen to affect the result, though the phase congruency is largest at the right place.



One interpretation of the measure is that since for small angles,  $\cos \theta = 1 - \theta^2$  then Equation 4.30 expresses the ratio of the magnitudes weighted by the variance of the difference to the summed magnitude of the components. There is certainly difficulty with this measure, apart from difficulty in implementation: it is sensitive to noise, as is any phase measure; it is not conditioned by the magnitude of a response (small responses are not discounted); and it is not well localised (the measure varies with the cosine of the difference in phase, not with the difference itself – though it does avoid discontinuity problems with direct use of angles). In fact, the phase congruency is directly proportional to the local energy [Venkatesh89], so an alternative approach is to search for maxima in the local energy. The notion of local energy allows us to compensate for the sensitivity to the detection of phase in noisy situations.

For these reasons, Kovessi developed a *wavelet*-based measure which improved performance, whilst accommodating noise. In basic form, phase congruency can be determined by convolving a set of wavelet filters with an image, and calculating the difference between the average filter response and the individual filter responses. The response of a (one-dimensional) signal  $I$  to a set

of wavelets at scale  $n$  is derived from the convolution of the cosine and sine wavelets (discussed in Section 2.7.3) denoted  $M_n^e$  and  $M_n^o$ , respectively

$$[e_n(x), o_n(x)] = [I(x) * M_n^e, I(x) * M_n^o] \quad (4.31)$$

to deliver the even and odd components at the  $n^{\text{th}}$  scale  $e_n(x)$  and  $o_n(x)$ , respectively. The amplitude of the transform result at this scale is the local energy

$$A_n(x) = \sqrt{e_n(x)^2 + o_n(x)^2} \quad (4.32)$$

At each point  $x$  we will have an array of vectors which correspond to each scale of the filter. Given that we are only interested in phase congruency that occurs over a wide range of frequencies (rather than just at a couple of scales), the set of wavelet filters needs to be designed so that adjacent components overlap. By summing the even and odd components we obtain

$$\begin{aligned} F(x) &= \sum_n e_n(x) \\ H(x) &= \sum_n o_n(x) \end{aligned} \quad (4.33)$$

and a measure of the total energy  $A$  as

$$\sum_n A_n(x) \approx \sum_n \sqrt{e_n(x)^2 + o_n(x)^2} \quad (4.34)$$

then a measure of phase congruency is

$$PC(x) = \frac{\sqrt{F(x)^2 + H(x)^2}}{\sum_n A_n(x) + \varepsilon} \quad (4.35)$$

where the addition of a small factor  $\varepsilon$  in the denominator avoids division by zero and any potential result when values of the numerator are very small. This gives a measure of phase congruency, which is essentially a measure of the local energy. Kovessi improved on this, improving on the response to noise, developing a measure which reflects the confidence that the signal is significant relative to the noise. Further, he considers in detail the frequency domain considerations, and its extension to two dimensions [Kovessi99]. For two-dimensional (image) analysis, given that phase congruency can be determined by convolving a set of wavelet filters with an image, and calculating the difference between the average filter response and the individual filter responses. The filters are constructed in the frequency domain by using complementary spreading functions; the filters must be constructed in the Fourier domain because the log-Gabor function has a singularity at zero frequency. In order to construct a filter with appropriate properties, a filter is constructed in a manner similar to the Gabor wavelet, but here in the frequency domain and using different functions. Following Kovessi's implementation, the first filter is a low-pass filter, here a Gaussian filter  $g$  with  $L$  different orientations

$$g(\theta, \theta_l) = \frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{(\theta - \theta_l)^2}{2\sigma_s^2}} \quad (4.36)$$

where  $\theta$  is the orientation,  $\sigma_s$  controls the spread about that orientation and  $\theta_l$  is the angle of local orientation focus. The other spreading function is a band-pass filter, here a log-Gabor filter  $lg$  with  $M$  different scales.

$$lg(\omega, \omega_m) = \begin{cases} 0 & \omega = 0 \\ \frac{1}{\sqrt{2\pi}\sigma_\beta} e^{-\frac{(\log(\omega/\omega_m))^2}{2(\log(\beta))^2}} & \omega \neq 0 \end{cases} \quad (4.37)$$

where  $\omega$  is the scale,  $\beta$  controls bandwidth at that scale and  $\omega$  is the centre frequency at that scale. The combination of these functions provides a two-dimensional filter  $l2Dg$  which can act at different scales and orientations.

$$l2Dg(\omega, \omega_m, \theta, \theta_l) = g(\theta, \theta_l) \times lg(\omega, \omega_m) \quad (4.38)$$

One measure of phase congruency based on the convolution of this filter with the image  $\mathbf{P}$  is derived by inverse Fourier transformation  $\mathfrak{F}^{-1}$  of the filter  $l2Dg$  (to yield a spatial domain operator) which is convolved as

$$S(m)_{x,y} = \mathfrak{F}^{-1}(l2Dg(\omega, \omega_m, \theta, \theta_l))_{x,y} * \mathbf{P}_{x,y} \quad (4.39)$$

to deliver the convolution result  $S$  at the  $m^{\text{th}}$  scale. The measure of phase congruency over the  $M$  scales is then

$$PC_{x,y} = \frac{\left| \sum_{m=1}^M S(m)_{x,y} \right|}{\sum_{m=1}^M |S(m)_{x,y}| + \varepsilon} \quad (4.40)$$

where the addition of a small factor  $\varepsilon$  again avoids division by zero and any potential result when values of  $S$  are very small. This gives a measure of phase congruency, but is certainly a bit of an ouch, especially as it still needs refinement.

Note that keywords re-occur within phase congruency: frequency domain, wavelets and convolution. By its nature, we are operating in the frequency domain and there is not enough room in this text, and it is inappropriate to the scope here, to expand further. Despite this, the performance of phase congruency certainly encourages its consideration, especially if local illumination is likely to vary and if a range of features is to be considered. It is derived by an alternative conceptual basis, and this gives different insight, let alone performance. Even better, there is a Matlab implementation available, for application to images - allowing you to replicate its excellent results. There has been further research, noting especially its extension in ultrasound image analysis [Mulet-Parada00] and its extension to spatiotemporal form [Myerscough04].

## 4.4 Localised Feature Extraction

There are two main areas covered here. The traditional approaches aim to derive local features by measuring specific image properties. The main target has been to estimate curvature: peaks of local curvature are corners and analysing an image by its corners is especially suited to image of man-made objects. The second area includes more modern approaches that improve performance by employing region or patch-based analysis. We shall start with the more established curvature based operators, before moving to the patch or region-based analysis.



#### 4.4.1 Detecting Image Curvature (Corner Extraction)

##### 4.4.1.1 Definition of Curvature

Edges are perhaps the low-level image features that are most obvious to human vision. They preserve significant features, so we can usually recognise what an image contains from its edge-detected version. However, there are other low-level features that can be used in computer vision. One important feature is *curvature*. Intuitively, we can consider curvature as the rate of change in edge direction. This rate of change characterises the points in a curve; points where the edge direction changes rapidly are *corners*, whereas points where there is little change in edge direction correspond to straight lines. Such extreme points are very useful for shape description and matching, since they represent significant information with reduced data.

Curvature is normally defined by considering a parametric form of a planar curve. The parametric contour  $v(t) = x(t)U_x + y(t)U_y$  describes the points in a continuous curve as the end points of the position vector. Here, the values of  $t$  define an arbitrary parameterisation, the unit vectors are again  $U_x = [1, 0]$  and  $U_y = [0, 1]$ . Changes in the position vector are given by the tangent vector function of the curve  $v(t)$ . That is,  $\dot{v}(t) = \dot{x}(t)U_x + \dot{y}(t)U_y$ . This vectorial expression has a simple intuitive meaning. If we think of the trace of the curve as the motion of a point and  $t$  is related to time, the tangent vector defines the instantaneous motion. At any moment, the point moves with a speed given by  $|\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}$  in the direction  $\phi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$ . The curvature at a point  $v(t)$  describes the changes in the direction  $\phi(t)$  with respect to changes in arc length. That is,

$$\kappa(t) = \frac{d\phi(t)}{ds} \quad (4.41)$$

where  $s$  is arc length, along the edge itself. Here  $\phi$  is the angle of the tangent to the curve. That is,  $\phi = \theta \pm 90^\circ$ , where  $\theta$  is the gradient direction defined in Equation 4.13. That is, if we apply an edge detector operator to an image, then we can compute  $\phi$  to obtain a normal direction for each point in a curve. The tangent to a curve is given by an orthogonal vector. Curvature is given with respect to arc length because a curve parameterised by arc length maintains a constant speed of motion. Thus, curvature represents changes in direction for constant displacements along the curve. By considering the chain rule, we have

$$\kappa(t) = \frac{d\phi(t)}{dt} \frac{dt}{ds} \quad (4.42)$$

The differential  $ds/dt$  defines the change in arc length with respect to the parameter  $t$ . If we again consider the curve as the motion of a point, this differential defines the instantaneous change in distance with respect to time. That is, the instantaneous speed. Thus,

$$ds/dt = |\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \quad (4.43)$$

and

$$dt/ds = 1/\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \quad (4.44)$$

By considering that  $\phi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$ , then the curvature at a point  $v(t)$  in Equation 4.42 is given by

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{(\dot{x}^2(t) + \dot{y}^2(t))^{3/2}} \quad (4.45)$$

This relationship is called the *curvature function* and it is the standard measure of curvature for *planar curves* [Apostol66]. An important feature of curvature is that it relates the derivative of a tangential vector to a normal vector. This can be explained by the simplified Serret-Frenet equations [Goetz70] as follows. We can express the tangential vector in polar form as

$$\dot{v}(t) = |\dot{v}(t)| \left( \cos(\phi(t)) + j\sin(\phi(t)) \right) \quad (4.46)$$

If the curve is parameterised by arc length, then  $|\dot{v}(t)|$  is constant. Thus, the derivative of a tangential vector is simply given by

$$\ddot{v}(t) = |\dot{v}(t)| \left( -\sin(\phi(t)) + j\cos(\phi(t)) \right) \frac{d\phi(t)}{dt} \quad (4.47)$$

Since we are using a normal parameterisation, then  $d\phi(t)/dt = d\phi(t)/ds$ . Thus, the tangential vector can be written as

$$\ddot{v}(t) = \kappa(t)\mathbf{n}(t) \quad (4.48)$$

where  $\mathbf{n}(t) = |\dot{v}(t)| \left( -\sin(\phi(t)) + j\cos(\phi(t)) \right)$  defines the direction of  $\ddot{v}(t)$  whilst the curvature  $\kappa(t)$  defines its modulus. The derivative of the normal vector is given by  $\dot{\mathbf{n}}(t) = |\dot{v}(t)| \left( -\cos(\phi(t)) - j\sin(\phi(t)) \right) \frac{d\phi(t)}{ds}$  and can be written as

$$\dot{\mathbf{n}}(t) = -\kappa(t)\dot{v}(t) \quad (4.49)$$

Clearly  $\mathbf{n}(t)$  is normal to  $\dot{v}(t)$ . Therefore, for each point in the curve, there is a pair of orthogonal vectors  $\dot{v}(t)$  and  $\mathbf{n}(t)$  whose moduli are proportionally related by the curvature.

Generally, the curvature of a parametric curve is computed by evaluating Equation 4.45. For a straight line, for example, the second derivatives  $\ddot{x}(t)$  and  $\ddot{y}(t)$  are zero, so the curvature function is nil. For a circle of radius  $r$ , we have that  $\dot{x}(t) = r \cos(t)$  and  $\dot{y}(t) = -r \sin(t)$ . Thus,  $\ddot{x}(t) = -r \sin(t)$ ,  $\ddot{y}(t) = -r \cos(t)$  and  $\kappa(t) = 1/r$ . However, for curves in digital images, the derivatives must be computed from discrete data. This can be done in four main ways. The most obvious approach is to calculate curvature by directly computing the difference between angular direction of successive edge pixels in a curve. A second approach is to derive a measure of curvature from changes in image intensity. Finally, a measure of curvature can be obtained by correlation.

#### 4.4.1.2 Computing Differences in Edge Direction

Perhaps the easier way to compute curvature in digital images is to measure the *angular change* along the curve's path. This approach was considered in early corner detection techniques [Bennett75][Groan78][Kitchen82] and it merely computes the difference in edge direction between connected pixels forming a discrete curve. That is, it approximates the derivative in Equation 4.41 as the difference between neighbouring pixels. As such, curvature is simply given by

$$k(t) = \phi_{t+1} - \phi_{t-1} \quad (4.50)$$

where the sequence  $\dots\phi_{t-1}, \phi_t, \phi_{t+1}, \phi_{t+2}, \dots$  represents the gradient direction of a sequence of pixels defining a curve segment. Gradient direction can be obtained as the angle given by an edge detector operator. Alternatively, it can be computed by considering the positions of pixels in the sequence.

That is, by defining  $\phi_t = (y_{t-1} - y_{t+1}) / (x_{t-1} - x_{t+1})$  where  $(x_t, y_t)$  denotes pixel  $t$  in the sequence. Since edge points are only defined at discrete points, this angle can only take eight values, so the computed curvature is very ragged. This can be smoothed by considering the difference in mean angular direction of  $n$  pixels on the leading and trailing curve segment. That is,

$$k_n(t) = \frac{1}{n} \sum_{i=1}^n \phi_{t+i} - \frac{1}{n} \sum_{i=-n}^{-1} \phi_{t+i} \quad (4.51)$$

The average also gives some immunity to noise and it can be replaced by a weighted average if Gaussian smoothing is required. The number of pixels considered, the value of  $n$ , defines a compromise between accuracy and noise sensitivity. Notice that filtering techniques may also be used to reduce the quantisation effect when angles are obtained by an edge detection operator. As we have already discussed, the level of filtering is related to the size of the template (as in Section 3.4.3).

In order to compute angular differences, we need to determine connected edges. Code 4.9 uses the data in the arrays `magnitude` and `angle` that contain the magnitude and angle of edges computed with the Canny edge operator. For each edge (i.e., a pixel with magnitude greater than one) the code fills the list `edgesNeighbours` with all the close edges. The curvature is computed by the average of the difference between the angle between the edge and the edges in its neighbour. The angle difference is computed by using the dot product of the vectors defining the edge direction.

```

for x,y in itertools.product(range(0, width), range(0, height)):
    # Edge
    if magnitude[y,x] > 0:
        # Consider neighbor edges
        edgesNeighbor = [ ]
        for wx,wy in itertools.product(range(-windowDelta, windowDelta+1),
                                       range(-windowDelta, windowDelta+1)):
            if magnitude[y+wy, x+wx] > 0 :
                edgesNeighbor.append((y+wy,x+wx))

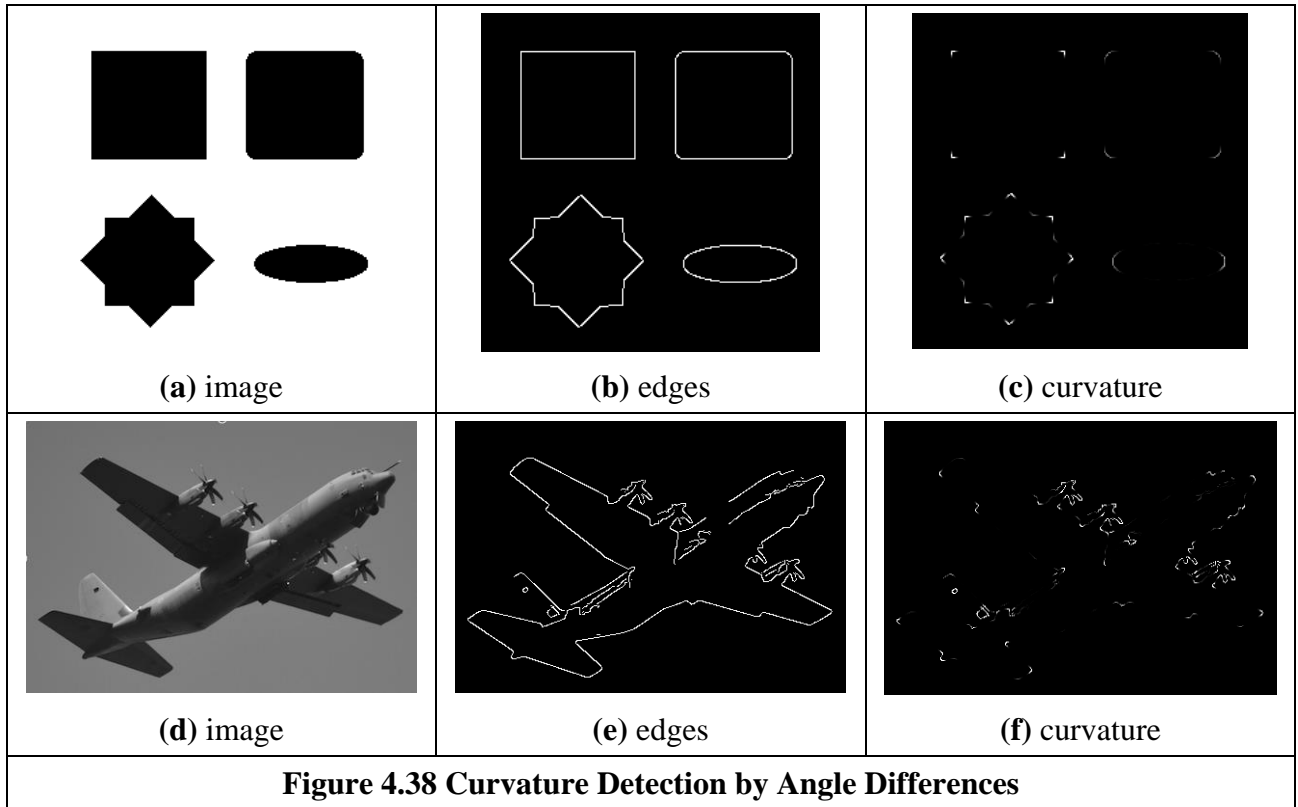
        # Use dot product to measure angle difference
        np = len(edgesNeighbor)
        for p in range(0, np):
            y1 = (edgesNeighbor[p])[0]
            x1 = (edgesNeighbor[p])[1]
            curvature[y,x] += 1.0 - (cos(angle[y1,x1]) * cos(angle[y,x])
                                   + sin(angle[y1,x1]) * sin(angle[y,x]))

        if np > 0:
            curvature[y,x] /= np

```

**Code 4.9 Detecting Curvature by Angle Differences**

The result of applying this type of curvature detection is shown in Figure 4.38. Here Figure 4.38(a) shows an image with regions whose borders have different curvature. Figure 4.38(d) shows an image of a real object. The edges computed by the Canny edge detector are shown in Figures 4.38(b) and (e). Figures 4.38(c) and (f) show the curvature obtained by computing the rate of change of edge direction. In these figures, curvature is defined only at the edge points. Here, by its formulation the measurement of curvature  $\mathcal{K}$  gives just a thin line of differences in edge direction which can be seen to track the perimeter points of the shapes (at points where there is measured curvature). The brightest points are those with greatest curvature. In order to show the results, we have scaled the curvature values to use 256 intensity values. The estimates of corner points could be obtained by a uniformly thresholded version of Figure 4.38(f), well in theory anyway!



Unfortunately, as can be seen, this approach does not provide reliable results. It is essentially a reformulation of a first order edge detection process and presupposes that the corner information lies within the threshold data (and uses no corner structure in detection). One of the major difficulties with this approach is that measurements of angle can be severely affected by quantisation error and accuracy is limited [Bennett75], a factor which will return to plague us later when we study methods for describing shapes.

#### 4.4.1.3 Measuring Curvature by Changes in Intensity (Differentiation)

As an alternative way of measuring curvature, we can derive the curvature as a function of changes in image intensity. This derivation can be based on the measure of angular changes in the discrete image. We can represent the direction at each image point as the function  $\phi'(x, y)$ . Thus, according to the definition of curvature, we should compute the change in these direction values normal to the image edge (i.e., along the curves in an image). The curve at an edge can be locally approximated by the points given by the parametric line defined by  $x(t) = x + t \cos(\phi'(x, y))$  and  $y(t) = y + t \sin(\phi'(x, y))$ . Thus, the curvature is given by the change in the function  $\phi'(x, y)$  with respect to  $t$ . That is,

$$\kappa_{\phi'}(x, y) = \frac{\partial \phi'(x, y)}{\partial t} = \frac{\partial \phi'(x, y)}{\partial x} \frac{\partial x(t)}{\partial t} + \frac{\partial \phi'(x, y)}{\partial y} \frac{\partial y(t)}{\partial t} \quad (4.52)$$

where  $\partial x(t)/\partial t = \cos(\phi')$  and  $\partial y(t)/\partial t = \sin(\phi')$ . By considering the definition of the gradient angle, we have that the normal tangent direction at a point in a line is given by  $\phi'(x, y) = \tan^{-1}(Mx/(-My))$ . From this geometry we can observe that

$$\cos(\phi') = -My/\sqrt{Mx^2 + My^2} \quad \text{and} \quad \sin(\phi') = Mx/\sqrt{Mx^2 + My^2} \quad (4.53)$$

By differentiation of  $\phi'(x, y)$  and by considering these definitions we obtain

$$\kappa_{\phi'}(x, y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}} \left\{ My^2 \frac{\partial Mx}{\partial x} - MxMy \frac{\partial My}{\partial x} + Mx^2 \frac{\partial My}{\partial y} - MxMy \frac{\partial Mx}{\partial y} \right\} \quad (4.54)$$

This defines a forward measure of curvature along the edge direction. We can actually use an alternative direction to the measure of curvature. We can differentiate backwards (in the direction of  $-\phi'(x, y)$ ) which gives  $\kappa_{-\phi'}(x, y)$ . In this case we consider that the curve is given by  $x(t) = x + t \cos(-\phi'(x, y))$  and  $y(t) = y + t \sin(-\phi'(x, y))$ . Thus,

$$\kappa_{-\phi'}(x, y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}} \left\{ My^2 \frac{\partial Mx}{\partial x} - MxMy \frac{\partial My}{\partial x} - Mx^2 \frac{\partial My}{\partial y} + MxMy \frac{\partial Mx}{\partial y} \right\} \quad (4.55)$$

Two further measures can be obtained by considering the forward and a backward differential along the normal. These differentials cannot be related to the actual definition of curvature, but can be explained intuitively. If we consider that curves are more than one pixel wide, differentiation along the edge will measure the difference between the gradient angle between interior and exterior borders of a wide curve. In theory, the tangent angle should be the same. However, in discrete images there is a change due to the measures in a window. If the curve is a straight line, then the interior and exterior borders are the same. Thus, gradient direction normal to the edge does not change locally. As we bend a straight line, we increase the difference between the curves defining the interior and exterior borders. Thus, we expect the measure of gradient direction to change. That is, if we differentiate along the normal direction, we maximise detection of gross curvature. The value  $\kappa_{\perp\phi'}(x, y)$  is obtained when  $x(t) = x + t \sin(\phi'(x, y))$  and  $y(t) = y + t \cos(\phi'(x, y))$ . In this case,

$$\kappa_{\perp\phi'}(x, y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}} \left\{ Mx^2 \frac{\partial My}{\partial x} - MxMy \frac{\partial My}{\partial x} - MxMy \frac{\partial My}{\partial y} + My^2 \frac{\partial Mx}{\partial y} \right\} \quad (4.56)$$

In a backward formulation along a normal direction to the edge, we obtain:

$$\kappa_{-\perp\phi'}(x, y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}} \left\{ -Mx^2 \frac{\partial My}{\partial x} + MxMy \frac{\partial Mx}{\partial x} - MxMy \frac{\partial My}{\partial y} + My^2 \frac{\partial Mx}{\partial y} \right\} \quad (4.57)$$

This was originally used by Kass [Kass88] as a means to detect *line terminations*, as part of a feature extraction scheme called snakes (active contours) which are covered in Chapter 6. Code 4.10 shows an implementation of the four measures of curvature. The arrays `mX` and `mY` store the gradient obtained by the convolutions of the original image with Sobel kernels in horizontal and vertical directions. The arrays `mXx`, `mXy`, `mYx` and `mYy` contain the convolutions of `mX` and `mY` with Sobel kernels. Figure 4.39 shows the form of the combined gradients where each combination detects changes in a different direction in the image. Code 4.10 combines the first and second order gradients according to *backward* and *forward* formulations along the normal and edge direction. The result is computed according to the selection of parameter `op`. The curvature value is only computed at edge pixels.

```

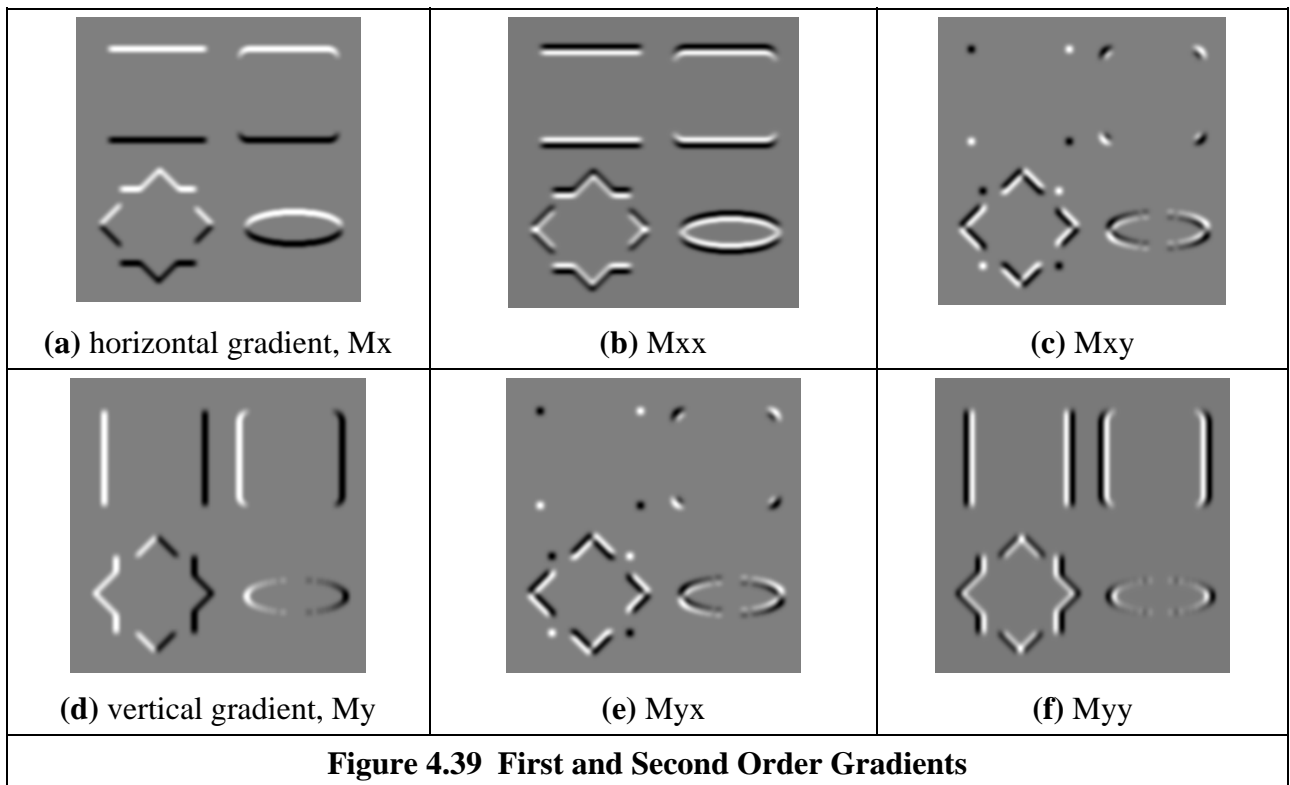
for x,y in itertools.product(range(0, width), range(0, height)):
    # If it is an edge
    if magnitude[y,x] > 0:
        Mx2,My2,MxMy = mX[y,x]*mX[y,x], mY[y,x]*mY[y,x], mX[y,x]*mY[y,x]

        if Mx2 + My2 !=0.0:
            p = 1.0/ pow((Mx2 + My2), 1.5)

            if op == "T":
                curvature[y,x] = p * (My2 * mXx[y,x] - MxMy * mYx[y,x] + \
                    Mx2 * mYy[y,x] - MxMy * mXy[y,x])
            if op == "TI":
                curvature[y,x] = p * (-My2 * mXx[y,x] + MxMy * mYx[y,x] - \
                    Mx2 * mYy[y,x] + MxMy * mXy[y,x])
            if op == "N":
                curvature[y,x] = p * (Mx2 * mYx[y,x] - MxMy * mYx[y,x] - \
                    MxMy * mYy[y,x] + My2 * mXx[y,x])
            if op == "NI":
                curvature[y,x] = p * (-Mx2 * mYx[y,x] + MxMy * mXx[y,x] + \
                    MxMy * mYy[y,x] - My2 * mXy[y,x])

        curvature[y,x] = fabs(curvature[y,x])
    
```

**Code 4.10 Curvature by measuring changes in intensity**



**Figure 4.39 First and Second Order Gradients**

Figure 4.40 shows the result for each operation for the synthetic image in Figure 4.38(a). We can see that the gradients locate edges and corners in different directions. That is, white and black alternates in each definition. However, all operations obtain maxima (or minima) where there is high curvature, so any of them can be used for curvature detection.

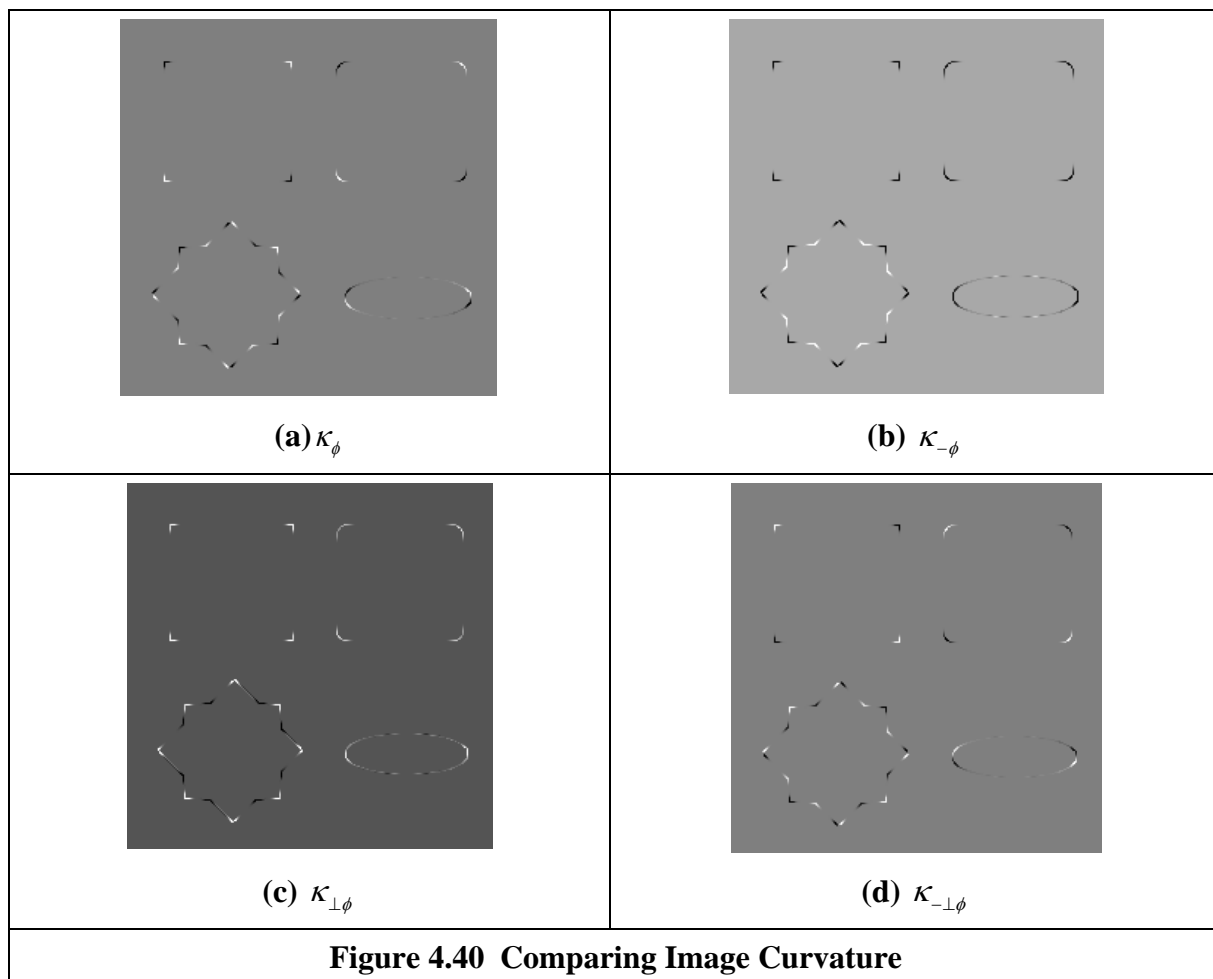
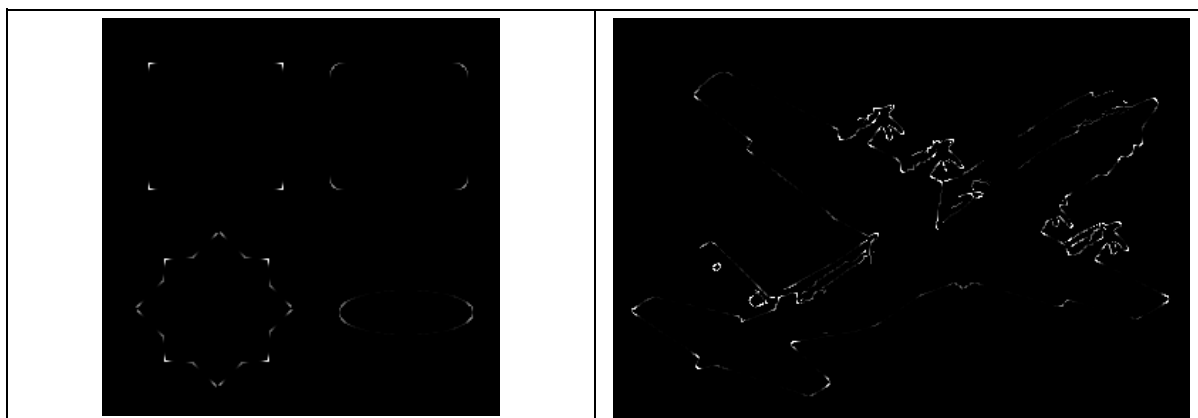


Figure 4.41 shows the curvature detected for the images shown in Figure 4.38. Curvature is obtained by the absolute value. In the figure, points where the curvature is large are highlighted. All functions obtain similar results and highlight features in the boundary. In general, we expect that the computation based on regions obtains better edges than when using angle differences (Figure 4.39) since convolution provides some filtering to the edges though there is little discernible performance. As the results in Figure 4.41 suggest, detecting curvature directly from an image is not totally reliable and further analysis shows that the computed values are just rough estimates of the actual curvature, however the results also show that these measures can effectively give strong evidence about the location of corners. The results are also very dependent on the size of the window used to compute the gradients and the size of the corners.



(a) Curvature magnitude $ \kappa_\phi $	(b) Curvature magnitude $ \kappa_\phi $
<b>Figure 4.41 Curvature Detection Operators</b>	

#### 4.4.1.4 Moravec and Harris Detectors

In the previous section, we measured curvature as the derivative of the function  $\phi(x, y)$  along the normal and edge directions. Alternatively, a measure of curvature can be obtained by considering changes along alternative directions in the image  $\mathbf{P}$  itself. This is the basic idea of *Moravec's corner detection operator*. This operator computes the average change in image intensity when a window is shifted in several directions. That is, for a pixel with coordinates  $(x, y)$ , and a window size of  $2w+1$  we have that

$$\mathbf{E}_{u,v}(x, y) = \sum_{i=-w}^w \sum_{j=-w}^w \left[ \mathbf{P}_{x+i,y+j} - \mathbf{P}_{x+i+u,y+j+v} \right]^2 \quad (4.58)$$

This equation approximates the *autocorrelation function* in the direction  $(u, v)$ . A measure of curvature is given by the minimum value of  $\mathbf{E}_{u,v}(x, y)$  obtained by considering the shifts  $(u, v)$  in the four main directions. That is, by  $(1, 0)$ ,  $(0, -1)$ ,  $(0, 1)$  and  $(-1, 0)$ . The minimum is chosen because it agrees with the following two observations. First, if the pixel is in an edge defining a straight line,  $\mathbf{E}_{u,v}(x, y)$  is small for a shift along the edge and large for a shift perpendicular to the edge. In this case, we should choose the small value since the curvature of the edge is small. Secondly, if the edge defines a corner, then all the shifts produce a large value. Thus, if we also chose the minimum, this value indicates high curvature. The main problem with this approach is that it considers only a small set of possible shifts. This problem is solved in the *Harris corner detector* [Harris88] by defining an analytic expression for the autocorrelation. This expression can be obtained by considering the local approximation of intensity changes.

We can consider that the points  $\mathbf{P}_{x+i,y+j}$  and  $\mathbf{P}_{x+i+u,y+j+v}$ , define a vector  $(u, v)$  in the image. Thus, in a similar fashion to the development given in Equation 4.58, the increment in the image function between the points can be approximated by the directional derivative  $u \partial \mathbf{P}_{x+i,y+j} / \partial x + v \partial \mathbf{P}_{x+i,y+j} / \partial y$ . Thus, the intensity at  $\mathbf{P}_{x+i+u,y+j+v}$  can be approximated as

$$\mathbf{P}_{x+i+u,y+j+v} = \mathbf{P}_{x+i,y+j} + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} u + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} v \quad (4.59)$$

This expression corresponds to the three first terms of the Taylor expansion around  $\mathbf{P}_{x+i,y+j}$  (an expansion to first-order). If we consider the approximation in Equation 4.58 we have that

$$\mathbf{E}_{u,v}(x, y) = \sum_{i=-w}^w \sum_{j=-w}^w \left[ \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} u + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} v \right]^2 \quad (4.60)$$

By expansion of the squared term (and since  $u$  and  $v$  are independent of the summations), we obtain,

$$\mathbf{E}_{u,v}(x, y) = A(x, y)u^2 + 2C(x, y)uv + B(x, y)v^2 \quad (4.61)$$

where



$$\begin{aligned}
 A(x, y) &= \sum_{i=-w}^w \sum_{j=-w}^w \left( \frac{\partial \mathbf{P}_{x+i, y+j}}{\partial x} \right)^2 & B(x, y) &= \sum_{i=-w}^w \sum_{j=-w}^w \left( \frac{\partial \mathbf{P}_{x+i, y+j}}{\partial y} \right)^2 \\
 C(x, y) &= \sum_{i=-w}^w \sum_{j=-w}^w \left( \frac{\partial \mathbf{P}_{x+i, y+j}}{\partial x} \right) \left( \frac{\partial \mathbf{P}_{x+i, y+j}}{\partial y} \right)
 \end{aligned} \tag{4.62}$$

That is, the summation of the squared components of the gradient direction for all the pixels in the window. In practice, this average can be weighted by a Gaussian function to make the measure less sensitive to noise (i.e. by filtering the image data). In order to measure the curvature at a point  $(x, y)$ , it is necessary to find the vector  $(u, v)$  that minimises  $\mathbf{E}_{u,v}(x, y)$  given in Equation 4.61. In a basic approach, we can recall that the minimum is obtained when the window is displaced in the direction of the edge. Thus, we can consider that  $u = \cos(\phi(x, y))$  and  $v = \sin(\phi(x, y))$ . These values were defined in Equation 4.53. Accordingly, the minima values that define curvature are given by

$$\kappa_{u,v}(x, y) = \min \mathbf{E}_{u,v}(x, y) = \frac{A(x, y)M_y^2 + 2C(x, y)M_xM_y + B(x, y)M_x^2}{M_x^2 + M_y^2} \tag{4.63}$$

In a more sophisticated approach, we can consider the form of the function  $\mathbf{E}_{u,v}(x, y)$ . We can observe that this is a quadratic function, so it has two principal axes. We can rotate the function such that its axes have the same direction that the axes of the co-ordinate system. That is, we rotate the function  $\mathbf{E}_{u,v}(x, y)$  to obtain

$$\mathbf{F}_{u,v}(x, y) = \alpha(x, y)^2 u^2 + \beta(x, y)^2 v^2 \tag{4.64}$$

The values of  $\alpha$  and  $\beta$  are proportional to the autocorrelation function along the principal axes. Accordingly, if the point  $(x, y)$  is in a region of constant intensity, we will have that both values are small. If the point defines a straight border in the image, then one value is large and the other is small. If the point defines an edge with high curvature, both values are large. Based on these observations a measure of curvature is defined as

$$\kappa_k(x, y) = \alpha\beta - k(\alpha + \beta)^2 \tag{4.65}$$

The first term in this equation makes the measure large when the values of  $\alpha$  and  $\beta$  increase. The second term is included to decrease the values in flat borders. The parameter  $k$  must be selected to control the sensitivity of the detector. The higher the value, the computed curvature will be more sensitive to changes in the image (and therefore to noise).

In practice, in order to compute  $\kappa_k(x, y)$  it is not necessary to compute explicitly the values of  $\alpha$  and  $\beta$ , but the curvature can be measured from the coefficient of the quadratic expression in Equation 4.61. This can be derived by considering the matrix forms of Equations 4.61 and 4.64. If we define the vector  $\mathbf{D}^T = [u, v]$ , then Equations 4.61 and 4.64 can be written as,

$$\mathbf{E}_{u,v}(x, y) = \mathbf{D}^T \mathbf{M} \mathbf{D} \quad \text{and} \quad \mathbf{F}_{u,v}(x, y) = \mathbf{D}^T \mathbf{Q} \mathbf{D} \tag{4.66}$$

where  $^T$  denotes the transpose and where

$$\mathbf{M} = \begin{bmatrix} A(x, y) & C(x, y) \\ C(x, y) & B(x, y) \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \tag{4.67}$$

In order to relate Equations 4.61 and 4.64, we consider that  $\mathbf{F}_{u,v}(x, y)$  is obtained by rotating  $\mathbf{E}_{u,v}(x, y)$  by a transformation  $\mathbf{R}$  that rotates the axis defined by  $\mathbf{D}$ . That is,

$$\mathbf{F}_{u,v}(x, y) = (\mathbf{RD})^T \mathbf{MRD} \quad (4.68)$$

This can be arranged as

$$\mathbf{F}_{u,v}(x, y) = \mathbf{D}^T \mathbf{R}^T \mathbf{MRD} \quad (4.69)$$

By comparison with Equation 4.66, we have that

$$\mathbf{Q} = \mathbf{R}^T \mathbf{MR} \quad (4.70)$$

This defines a well-known equation of linear algebra and it means that  $\mathbf{Q}$  is an orthogonal decomposition of  $\mathbf{M}$ . The diagonal elements of  $\mathbf{Q}$  are called the eigenvalues. We can use Equation 4.70 to obtain the value of  $\alpha\beta$  which defines the first term in Equation 4.65 by considering the determinant of the matrices. That is,  $\det(\mathbf{Q}) = \det(\mathbf{R}^T) \det(\mathbf{M}) \det(\mathbf{R})$ . Since  $\mathbf{R}$  is a rotation matrix  $\det(\mathbf{R}^T) \det(\mathbf{R}) = 1$ , thus

$$\alpha\beta = A(x, y)B(x, y) - C(x, y)^2 \quad (4.71)$$

which defines the first term in Equation 4.65. The second term can be obtained by taking the trace of the matrices on each side of this equation. Thus, we have that

$$\alpha + \beta = A(x, y) + B(x, y) \quad (4.72)$$

We can also use Equation 4.70 to obtain the value of  $\alpha + \beta$  which defines the first term in Equation 4.65. By taking the trace of the matrices in each side of this equation, we have that

$$\kappa_k(x, y) = A(x, y)B(x, y) - C(x, y)^2 - k(A(x, y) + B(x, y))^2 \quad (4.73)$$

Code 4.11 shows an implementation for Equations 4.64 and 4.73. The equation to be used is selected by the `op` parameter. Curvature is only computed at edge points. That is, at pixels whose edge magnitude is different from zero after applying non-maximum suppression. The first part of the code computes the coefficients of the matrix  $\mathbf{M}$ . Then, these values are used in the curvature computation.

```

for x,y in itertools.product(range(0, width), range(0, height)):
    # If it is an edge
    if magnitude[y,x] > 0:
        A, B, C = 0.0, 0.0, 0.0
        for wx,wy in itertools.product(range(0, kernelSize), range(0, kernelSize)):
            posY = y + wy - kernelCentre
            posX = x + wx - kernelCentre

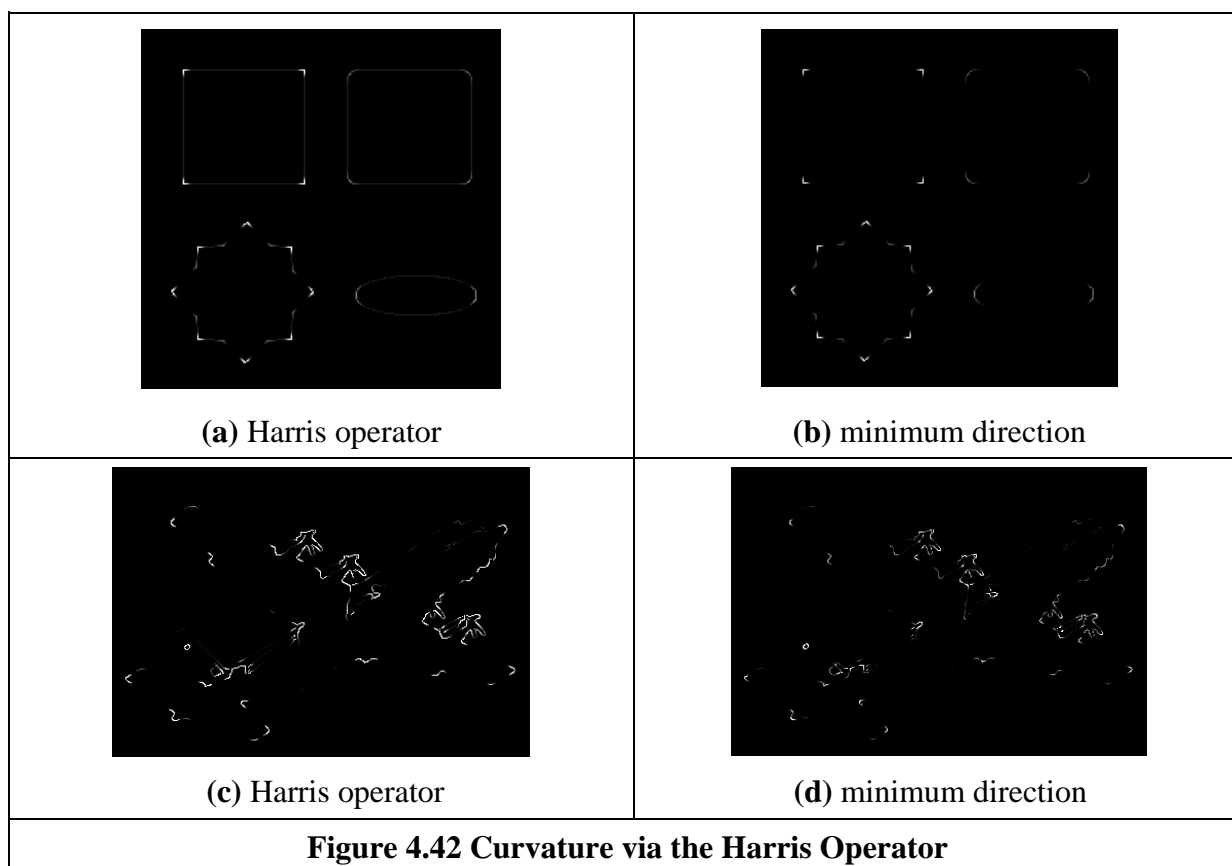
            if posY > -1 and posY < height and posX > -1 and posX < width:
                A += mX[poSY,poSX] * mX[poSY,poSX]
                B += mY[poSY,poSX] * mY[poSY,poSX]
                C += mX[poSY,poSX] * mY[poSY,poSX]

        if op == "H":
            curvature[y,x] = (A * B) - (C * C) - (k * ((A+B) * (A+B)))
        if op == "M":
            d = mX[y,x] * mX[y,x] + mY[y,x] * mY[y,x]
            if d != 0.0:
                curvature[y,x] = (A * mY[y,x] * mY[y,x] -
                    2.0 * C * mX[y,x] * mY[y,x] +
                    B * mX[y,x] * mX[y,x]) / d

```

<b>Code 4.11 Harris Corner Detector</b>
---

Figure 4.42 shows the results of computing curvature using this implementation. The results are capable of showing the different curvature values in the border. We can observe that the Harris operator produces more contrast between lines with low and high curvature than when curvature is computed by the evaluating the minimum direction. The reason is the inclusion of the second term in Equation 4.73. In general, the measure of correlation is not only useful for computing curvature, but this technique has much wider application in finding points when matching pairs of images.



#### 4.4.1.5 Further Reading on Curvature

Many of the arguments earlier advanced on extensions to edge detection in Section 4.2 apply to corner detection as well, so the same advice applies. There is much less attention paid by established textbooks to corner detection though [Davies05] devotes a Chapter to the topic. Van Otterloo's fine book on shape analysis [VanOtterloo91] contains a detailed analysis of measurement of (planar) curvature.

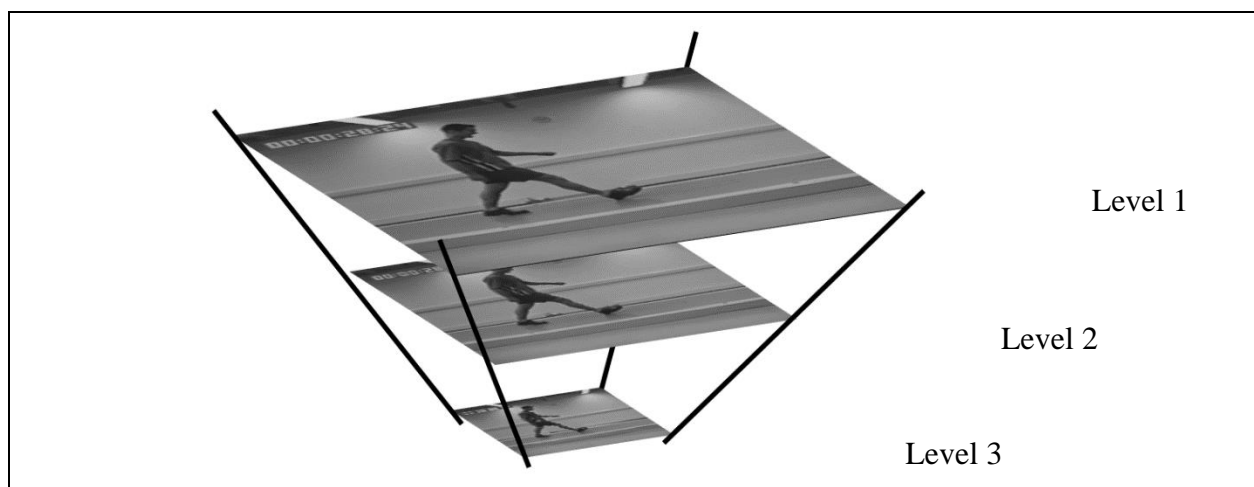
There are other important issues in corner detection. It has been suggested that corner extraction can be augmented by local knowledge to improve performance [Rosin96]. There are actually many other corner detection schemes, each offering different attributes though with differing penalties. Important work has focused on characterising shapes using corners. In a scheme analogous to the primal sketch introduced earlier, there is a *curvature primal sketch* [Asada86], which includes a set of primitive parameterised curvature discontinuities (such as termination and joining points). There are many other approaches: one (natural) suggestion is to define a corner as the intersection between two lines, this requires a process to find the lines; other techniques use methods that describe shape variation to find corners. We commented that filtering techniques can be included to improve the

detection process, however filtering can also be used to obtain a representation with multiple details. This representation is very useful to shape characterisation. A *curvature scale space* was developed [Mokhtarian86, Mokhtarian03] to give a compact way of representing shapes, and at different scales, from coarse (low-level) to fine (detail) and with ability to handle appearance transformations.

#### 4.4.2 Feature Point Detection; Region/Patch Analysis

The modern approaches to local feature extraction aim to relieve some of the constraints on the earlier methods of localised feature extraction. This allows for the inclusion of scale: an object can be recognised irrespective of its apparent size. The object might also be characterised by a collection of points, and this allows for recognition where there has been change in the viewing arrangement (in a planar image an object viewed from a different angle will appear different, but points which represent it still appear in a similar arrangement). Using arrangements of points also allows for recognition where some of the image points have been obscured (because the image contains clutter or noise). In this way, we can achieve a description which allows for object or scene recognition direct from the image itself, by exploiting local neighbourhood properties.

The newer techniques depend on the notion of scale space: features of interest are those which persist over selected scales. The scale space is defined by images which are successively smoothed by the Gaussian filter, as in Equation 3.26, and then subsampled to form an *image pyramid* at different scales, illustrated in Figure 4.43 for three levels of resolution. There are approaches which exploit structure within the scale space to improve speed, as we shall find.



**Figure 4.43 Illustrating Scale Space**

##### 4.4.2.1 Scale Invariant Feature Transform (SIFT)

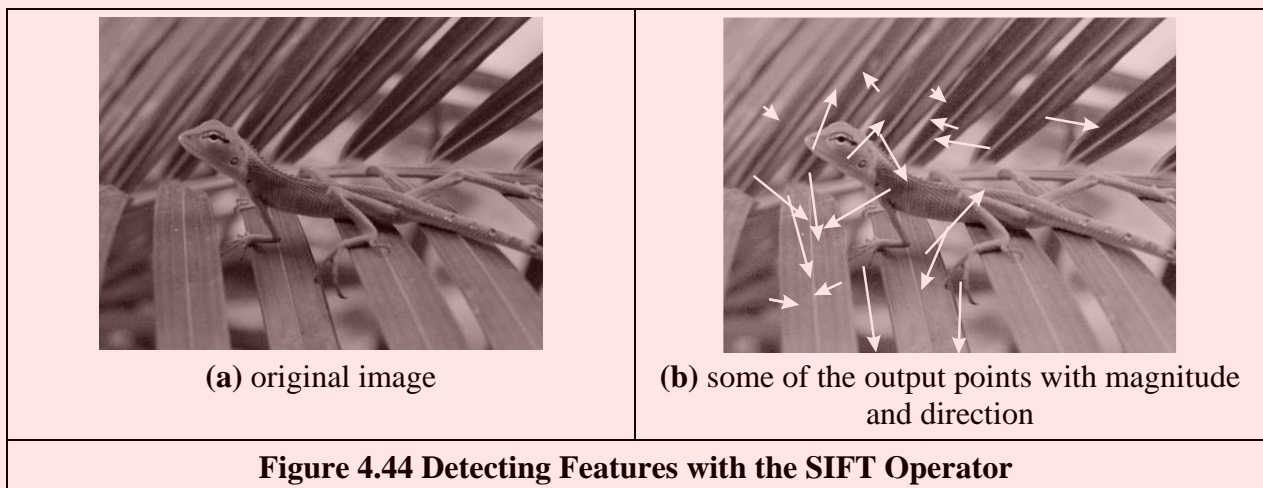
The *Scale Invariant Feature Transform (SIFT)* [Lowe99, Lowe04] aims to resolve many of the practical problems in low-level feature extraction and their use in matching images. The earlier Harris operator is sensitive to changes in image scale and as such is unsuited to matching images of differing size. The SIFT transform actually involves two stages: feature extraction and description. The description stage concerns use of the low-level features in object matching, and this will be considered later. Low-level feature extraction within the SIFT approach selects salient features in a manner invariant to image scale (feature size), and rotation and with partially

John Hare

invariance to change in illumination. Further, the formulation reduces the probability of poor extraction due to occlusion clutter and noise. Further, it shows how many of the techniques considered previously can be combined and capitalised on, to good effect.

First, the difference of Gaussians operator is applied to an image to identify features of potential interest. The formulation aims to ensure that feature selection does not depend on feature size (scale) or orientation. The features are then analysed to determine location and scale before the orientation is determined by local gradient direction. Finally, the features are transformed into a representation that can handle variation in illumination and local shape distortion. Essentially, the operator uses local information to refine the information delivered by standard operators. The detail of the operations is best left to the source material [Lowe99, Lowe04] for it is beyond the level or purpose here. As such, we shall concentrate on principle only.

Some features detected for an image are illustrated in Figure 4.44. Here, the major features detected are shown by white lines where the length reflects magnitude, and the direction reflects the feature's orientation. These are the major features which include the head of the lizard and some leaves. The minor features are the smaller white lines: the ones shown here are concentrated around background features. In the full set of features detected at all scales in this image, there are many more of the minor features, concentrated particularly in the textured regions of the image (Figure 4.45). Later we shall see how this can be used within shape extraction, but our purpose here is the basic low-level features.



**Figure 4.44 Detecting Features with the SIFT Operator**

In the first stage, the difference of Gaussians for an image  $\mathbf{P}$  is computed in the manner of Equation 4.28 as

$$\begin{aligned}
 D(x, y, \sigma) &= (g(x, y, k\sigma) - g(x, y, \sigma)) * \mathbf{P} \\
 &= L(x, y, k\sigma) - L(x, y, \sigma)
 \end{aligned}
 \tag{4.74}$$

The function  $L$  is actually a scale space function which can be used to define smoothed images at different scales. Rather than any difficulty in locating zero-crossing points, the features are the maxima and minima of the function. Candidate keypoints are then determined by comparing each point in the function with its immediate neighbours. The process then proceeds to analysis between the levels of scale, given appropriate sampling of the scale space. This then implies comparing a point with its eight neighbours at that scale and with the nine neighbours in each of the adjacent scales, to determine whether it is a minimum or maximum, as well as image resampling to ensure comparison between the different scales.

In order to filter the candidate points to reject those which are the result of low local contrast (low edge strength) or an edge, a function is

John Hare

which are poorly localised along derived by local curve fitting

John Hare

which indicates local edge strength and stability as well as location. Uniform thresholding then removes the keypoints with low contrast. Those that have poor localisation, i.e. their position is likely to be influenced by noise, can be filtered by considering the ratio of curvature along an edge to that perpendicular to it, in a manner following the Harris operator in Section 4.4.1.4, by thresholding the ratio of Equations 4.71 and 4.72,

In order to characterise the filtered keypoint features at each scale, the gradient magnitude is calculated in exactly the manner of Equations. 4.12 and 4.13 as

$$M_{SIFT}(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (4.75)$$

$$\theta_{SIFT}(x, y) = \tan^{-1} \left( \frac{L(x, y+1) - L(x, y-1)}{(L(x+1, y) - L(x-1, y))} \right) \quad (4.76)$$

The peak of the histogram of the orientations around a keypoint is then selected as the local direction of the feature. This can be used to derive a canonical orientation, so that the resulting descriptors are invariant with rotation. As such, this contributes to the process which aims to reduce sensitivity to camera viewpoint and to nonlinear change in image brightness (linear changes are removed by the gradient operations) by analysing regions in the locality of the selected viewpoint. The main description [Lowe04] considers the technique's basis in much greater detail, and outlines factors important to its performance such as the need for sampling and performance in noise.

As shown in Figure 4.45, the technique can certainly operate well, and scale is illustrated by applying the operator to the original image and to one at half the resolution. In all, 601 key points are determined in the original resolution image and 320 key points at half the resolution. By inspection, the major features are retained across scales (a lot of minor regions in the leaves disappear at lower resolution), as expected. Alternatively the features can of course be filtered further by magnitude, or even direction (if appropriate). If you want more than results to convince you, implementations are available for Windows and Linux (<https://www.cs.ubc.ca/~lowe/keypoints/> and some of the software sites noted in Table 1.2) – a feast for any developer. These images were derived by using `siftWin32`, version 4.

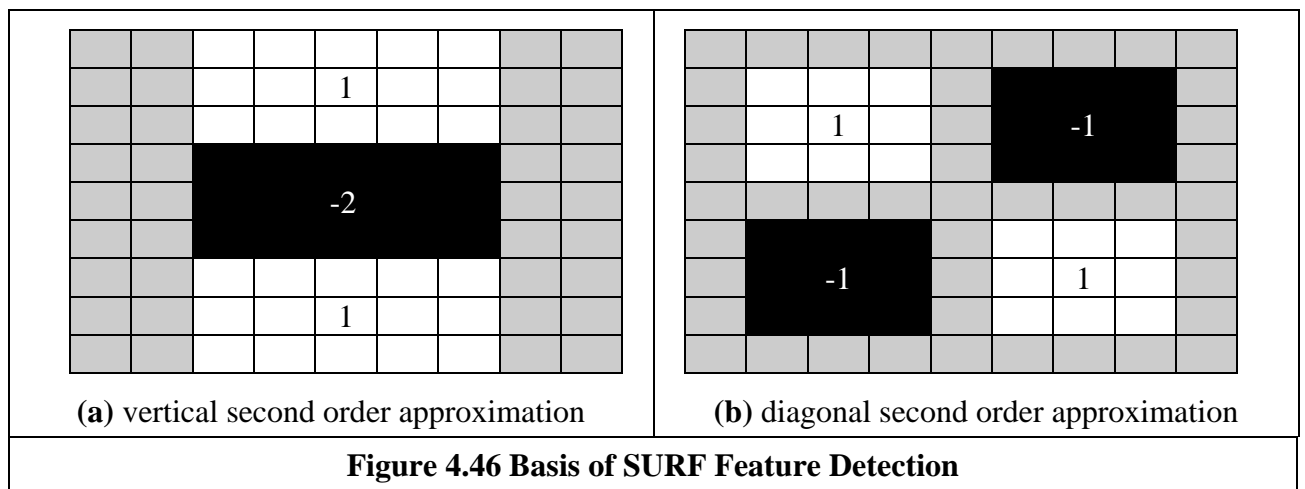
Note that description is inherent in the process - the standard SIFT keypoint descriptor is created by sampling the magnitudes and orientations of the image gradient in the region of the keypoint. An array of histograms, each with orientation bins, captures the rough spatial structure of the patch. This results in a vector which was later compressed by using PCA [Ke04] to determine the most salient features. Clearly this allows for faster matching than the original SIFT formulation, but the improvement in performance was later doubted [Mikolajczyk05].



Figure 4.45 SIFT Feature Detection at Different Scales

4.4.2.2 Speeded Up Robust Features (SURF)

The central property exploited within SIFT is the use of difference of Gaussians to determine local features. In a relationship similar to the one between first- and second-order edge detection, the *Speeded Up Robust Features (SURF)* approach [Bay06, Bay08] employs approximations to second order edge detection, at different scales. The basis of the SURF operator is to use the integral image approach of Section 2.7.3.2 to provide an efficient means to compute approximations of second order differencing, shown in Figure 4.46. These are the approximations for a Laplacian of Gaussian operator with  $\sigma = 1.2$  and represent the finest scale in the SURF operator. Other approximations can be derived for larger scales, since the operator – like SIFT – considers features which persist over scale space. The scale space can be derived by upscaling the approximations (wavelets) using larger templates which gives for faster execution than the use of smoothing and resampling in an image to form a pyramidal structure of different scales, which is more usual in scale-space approaches.



By the Taylor expansion of the image brightness (Equation 4.59) we can form a (Hessian) matrix  $\mathbf{M}$  (Equation 4.67) from which the maxima are used to derive the features. This is

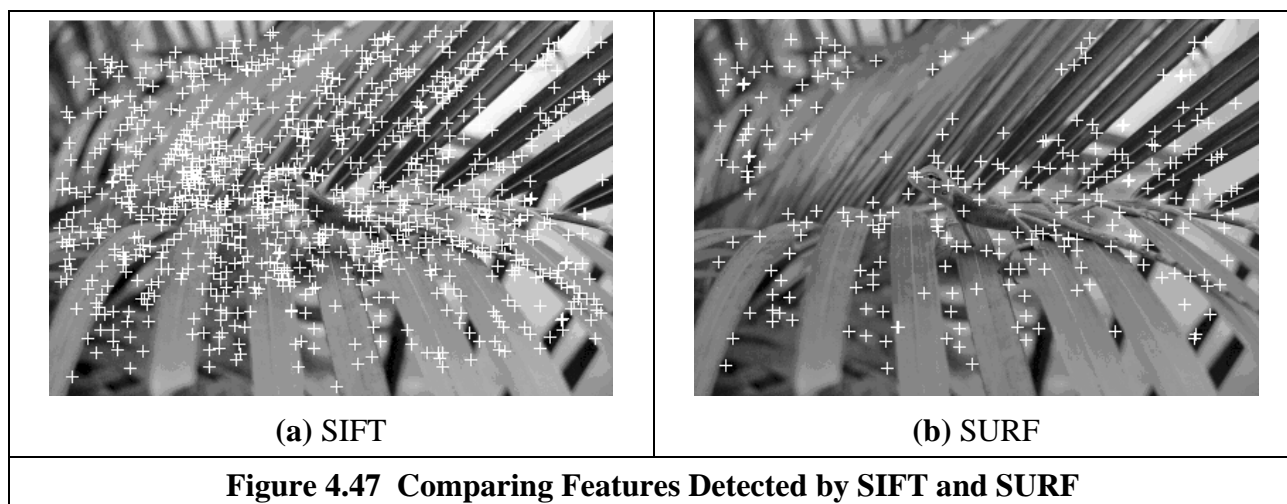
$$\det(\mathbf{M}) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix} = L_{xx}L_{yy} - w \cdot L_{xy}^2 \quad (4.77)$$

where the terms in  $\mathbf{M}$  arise from the convolution of a second order derivative of Gaussian with the image information as

$$L_{xx} = \frac{\partial^2 (g(x, y, \sigma))}{\partial x^2} * \mathbf{P}_{x,y} \quad L_{xy} = \frac{\partial^2 (g(x, y, \sigma))}{\partial x \partial y} * \mathbf{P}_{x,y} \quad L_{yy} = \frac{\partial^2 (g(x, y, \sigma))}{\partial y^2} * \mathbf{P}_{x,y} \quad (4.78)$$

and where  $w$  is carefully chosen to balance the components of the equation. To localise interest points in the image and over scales, nonmaximum suppression is applied in a  $3 \times 3 \times 3$  neighbourhood. The maxima of the determinant of the Hessian matrix are then interpolated in scale space and in image space and described by orientations derived using the vertical and horizontal Haar wavelets described earlier (Section 2.7.3.2). Not that there is an emphasis on speed of execution, as well as on performance attributes, and so the generation of the templates to achieve scale space, the factor  $w$ , the interpolation operation to derive features and then their description are achieved using optimised processes. The developers have provided downloads for evaluation of SURF from [www.vision.ee.ethz.ch/~surf/](http://www.vision.ee.ethz.ch/~surf/). The performance of the operator is illustrated in

Figure 4.47 showing the positions of the detected points for SIFT and for SURF. This shows that SURF can deliver fewer features, and which persist (and hence can be faster) whereas SIFT can provide more features (and be slower). As ever, choice depends on application – both techniques are available for evaluation and there are public domain implementations. The *Binary Robust Independent Elementary Features* (BRIEF) approach simply uses intensity comparison as a detector [Calonder10], which is innately simple though with proven capability, especially speed as it is very fast.



**Figure 4.47 Comparing Features Detected by SIFT and SURF**

#### 4.4.2.3 FAST, ORB, FREAK, LOCKY and other keypoint detectors

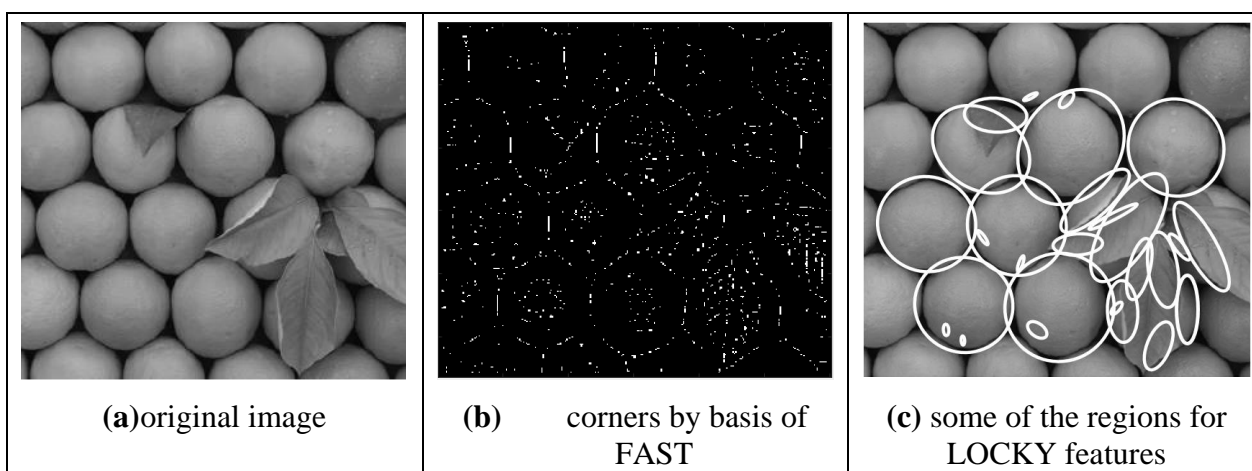
The need for fast interest point detection is manifest in the application of computer vision in *Content Based Image Retrieval (CBIR)* in robots and autonomous driving, and so the area has seen much attention. As illustrated in Figure 4.48 the aim is for matching: in autonomous driving one seeks a match from one image to the next; in face biometrics the target is points which do not change when the face changes in appearance, either by expression or by pose. The features are those which describe the underlying shape, and can be matched to each other. One basic tenet is whether the features are points-corners, or blobs (shown in Figure 4.49), and whether the features change with change in the image itself. If the image is moved (along either axis) and the features remain the same, this represents translation invariance. (The magnitude of the Fourier transform was previously shown to be shift invariant.) Then there is rotation, and then change in scale in the two axis directions. If an operator is invariant to change in translation, rotation and scale it is termed affine invariant and these points are termed keypoints. Not all the matched points are shown in Figure 4.49, just a subset of those automatically selected.





**Figure 4.48 Matching by Keypoint Detection**

The *Features from Accelerated Segment Test (FAST)* approach [Rosten06] is based on simple tests on a set of sixteen points centred on a pixel of interest with coordinates  $x, y$ . The points are arranged to be contiguous along a (discrete) circle and in principle if the brightness of all points on the circle exceed that of  $I(x, y)$  then the location  $x, y$  is labelled as a feature point. In more detail, given a threshold  $t$ , in a manner similar to hysteresis thresholding if three of the four points at compass directions (N, E, S and W) exceed  $I(x, y) + t$  or three of the four compass points are less than  $I(x, y) - t$  the point is possibly a feature point and the remaining twelve points are tested to determine if their brightness exceeds the centre point. Conversely if three of the four compass points do not exceed  $I(x, y) + t$  nor are three of the four compass points less than  $I(x, y) - t$  the point cannot be a feature point and the test is not conducted. The basis of the technique is fast by its use of comparisons. As can be observed in Figure 4.49(b), it offers good performance but relies on choice of a threshold and lacks robustness in noise and capability for extension to larger templates and so machine learning was deployed to improve its performance.



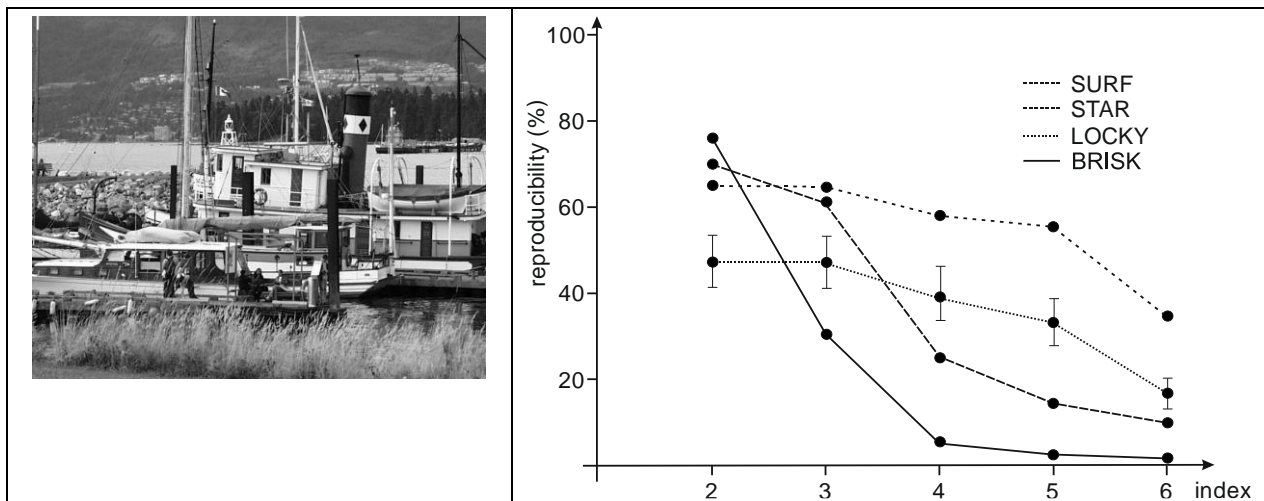
**Figure 4.49 Detecting Features by FAST and LOCKY**

A potential limitation of the FAST operator (when used for matching) is that it tests for existence and does not include orientation – unlike the SURF and SIFT operators. This limitation was alleviated by the *ORB* operator [Rublee11] (where the acronym derives from *Oriented FAST and Rotated BRIEF*, the latter being a descriptive operator) which thus includes rotation invariance. Since FAST does not produce a measure of cornerness, and can have large responses along edges, as to be seen in Figure 4.49(b), potential responses are filtered by shape measures. The basis of the filter is shape measures called moments, and these will be covered in Chapter 7. The extension of ORB to include descriptive capability is to be found in Chapter 5. In general, the approaches are known as binary descriptors since a binary string results from a sequence of comparisons. In ORB, pairs of points are learned to build the final descriptor.

Both FAST and ORB illustrate the nature of advanced operators, which employ many of the basic techniques used in Computer Vision, which are in shape extraction, description and machine learning. This illustrates a computer vision textbook author’s difficulty since one writes a book to be read with linear progression, so one resorts (as we do) to forward references where necessary.

The *Fast Retina Keypoint (FREAK)* operator [Alahi12] was inspired by analysis of the retina in the human visual system. The descriptor is built using intensity comparisons between regions which are smoothed using a Gaussian kernel. The regions are large at the periphery of a point of interest and much smaller when closer to it (a coarse-to-fine strategy), in a manner similar to the retina’s receptive field. Then, pairs of points are selected by variability, leading to a small and effective set of points. A more recent approach, the *Locally Contrasting Keypoints (LOCKY)* are features derived from a brightness clustering transform [Lomeli16] and can be thought as a coarse-to-fine search through scale spaces for the true derivative of the image. The brightness clustering uses a set of randomly positioned and (binary) sized rectangles which accumulate brightness using the integral image. The parameters to be chosen are the size of the rectangles and the number of votes to be accumulated. After thresholding the accumulated points the resulting image is thresholded and the LOCKY features are derived from the shapes of the connected components.

There are many more operators, each with particular advantage. E.g. the *Binary Robust Invariant Scalable Keypoints (BRISK)* operator [Leutenegger11] uses a corner detector and then detects keypoints in scale-space, *GIST* [Oliva06] employs Gabor wavelets aiming to describe scenes, and *STAR* [Agrawal08] uses a form of Haar wavelets via an integral image. By virtue for the need to find and match objects in images, this research is unlikely to cease with an optimum operator. Now, much research focusses on *Convolutional Neural Networks* [Zheng18] and this will be considered later in Chapter 12.



(a) boat reference image	(b) variation in reproducibility [Lomeli16]
<b>Figure 4.50 On the Performance of Interest Point Detectors [Lomeli16]</b>	

The Oxford affine-covariant regions dataset [Mikolajczyk05] is an established dataset for evaluating feature detection. It is made up from eight sequences which comprise six images, each with increasing image transformations including change in illumination, perspective, blurring, (jpeg) compression and a combination of scale and rotation. A measure of repeatability is formed by projecting the detected features onto the same basis as the reference image (say, the first image of the sequence) using a homography matrix. The repeatability measure is one of how well the features detected on the images overlap and it is preferable to achieve high repeatability. Using the correspondences with 40% overlap, Figure 4.50 shows the reduction in repeatability for change in image appearance (the index). Comparison has been chosen to show no particular advantage; each technique has proponents (and antagonists!). As LOCKY is non deterministic, it is shown with error bars (which are always welcome in any evaluation). One might prefer the consistency of SURF, or the performance of BRISK. LOCKY appears to be as fast as the STAR operator and faster than other approaches. But this is just one image, one series of (mainly OpenCV) implementations, and one evaluation methodology: there are many variations to this theme. Another comparison [Mukherjee15] highlights the all-round performance of ORB, SIFT and BRIEF, and is a good place to look. As ever, there is no panacea: techniques fit applications and not the other way round.

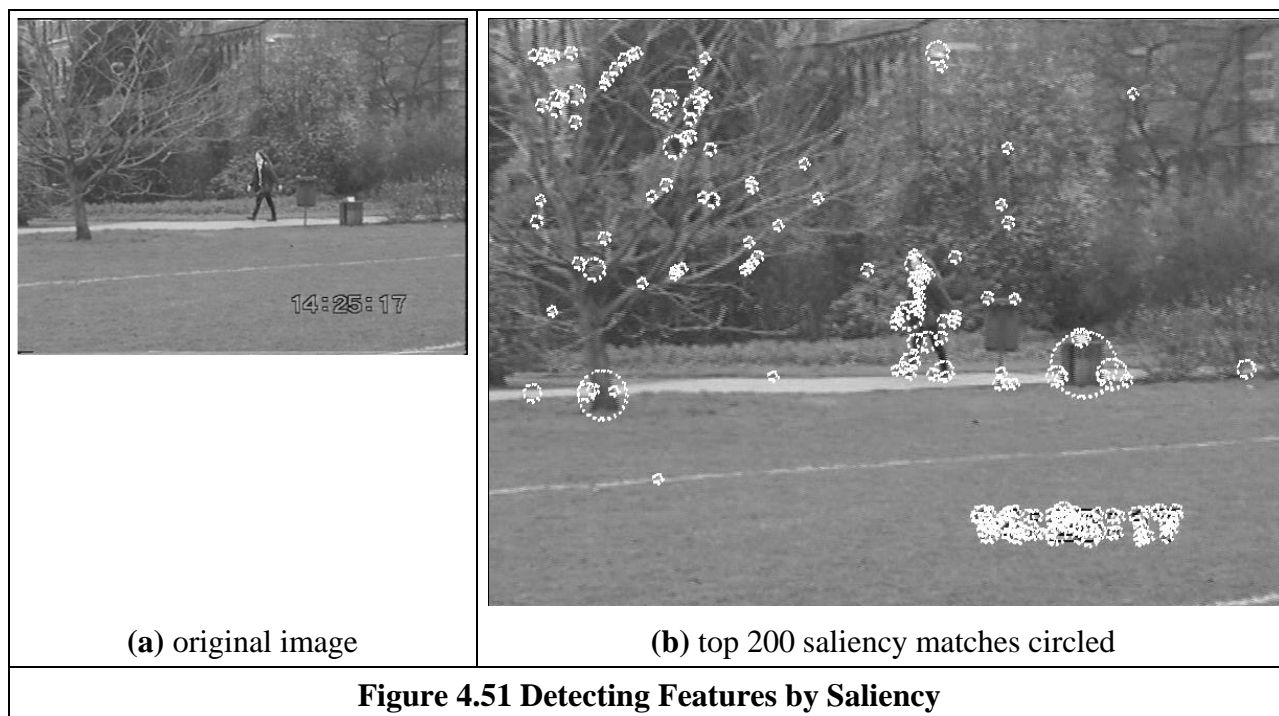
#### 4.4.2.4 Other Techniques and Performance Issues

There was a comprehensive *performance review* [Mikolajczyk05], comparing basic operators. The techniques which were compared included: SIFT; differential derivatives by differentiation; cross correlation for matching; and a gradient location and orientation based histogram (an extension to SIFT, which performed well). The criterion used for evaluation concerned the number of correct matches, and the number of false matches, between feature points selected by the techniques. The matching process was between an original image and one of the same scene when subject to one of six image transformations. The image transformations covered practical effects that can change image appearance and were: rotation; scale change; viewpoint change; image blur; JPEG compression; and illumination. For some of these there were two scene types available, which allowed for separation of understanding of scene type and transformation. The study observed that, within its analysis, “the SIFT-based descriptors perform best” but it is of course a complex topic and selection of technique is often application dependent. Note that there is further interest in performance evaluation, and in invariance to higher order changes in viewing geometry, such as invariance to affine and projective transformation. There are other comparisons available, either with new operators or in new applications and there (inevitably) are faster implementations too. A later survey covered the field in more detail [Tuyltelaars07], concerning principle and performance analysis. More recently, there has been much interest including deep learning for speed and accuracy. Given the continuing interest in CBIR, attention has turned to that. Note that CBIR includes detection and description, and the descriptions are covered in the next Chapter. Chapter 12 covers deep learning in particular, and it is worth noting here that there has been a recent survey [Zheng18], covering the development of instance-level image retrieval (where a query image is matched to target images) and of its relationship with deep learning. In the context of this Chapter, this concerns SIFT-based detection methods.

### 4.4.3 Saliency

#### 4.4.3.1 Basic Saliency

An early *saliency* operator [Kadir01] was also motivated by the need to extract robust and relevant features. In the approach, regions are considered salient if they are simultaneously unpredictable both in some feature and scale-space. Unpredictability (rarity) is determined in a statistical sense, generating a space of saliency values over position and scale, as a basis for later understanding. The technique aims to be a generic approach to scale and saliency compared to conventional methods, because both are defined independent of a particular basis morphology - meaning that it is not based on a particular geometric feature like a blob, edge or corner. The technique operates by determining the entropy (a measure of rarity) within patches at scales of interest and the saliency is a weighted summation of where the entropy peaks. The method has practical capability in that it can be made invariant to rotation, translation, non-uniform scaling, and uniform intensity variations and robust to small changes in viewpoint. An example result of processing the image in Figure 4.51(a) is shown in Figure 4.51(b) where the 200 most salient points are shown circled, and the radius of the circle is indicative of the scale. Many of the points are around the walking subject and others highlight significant features in the background, such as the waste bins, the tree or the time index. An example use of saliency was within an approach to learn and recognise object class models (such as faces, cars, or animals) from unlabelled and unsegmented cluttered scenes, irrespective of their overall size [Fergus03]. For further study and application, descriptions and Matlab binaries are available from Kadir's website (<http://www.robots.ox.ac.uk/~timork/>).



#### 4.4.3.2 Context Aware Saliency

There has been quite a surge of interest in saliency detection since the early works. The *context aware saliency* is grounded in psychology [Goferman12] (learning from human vision is invariably a good idea). The approach uses colour images, and the colour representation is the CIE Lab (to be covered later in Chapter 11) as it is known to be the closest system to human colour vision. The operator generates maps of saliency. In its basis, the operator uses distance which is oft a cue for

saliency in human vision. A point in the background of an image is likely to have similarity to points which are both near and far away. In contrast a point in the foreground, or a point of interest, is likely to be near another point of interest. Noting that a selection of distance measures is covered later in Chapter 12, the Euclidian distance  $d_p$  between the positions of two points  $p$  and  $q$  is

$$d_p(p, q) = \sqrt{(x(p) - x(q))^2 + (y(p) - y(q))^2} \quad (4.79)$$

which is normalised by the image dimensions. Similarly, in terms of colour, a point is likely to be salient when the difference in colour is large. The Euclidean distance in colour  $d_c$  between the two points is

$$d_c(p, q) = \sqrt{\sum_{cp=1}^{N_p} (\text{plane}_{cp}(p) - \text{plane}_{cp}(q))^2} \quad (4.80)$$

where  $N_p$  is the number of colour planes (usually 3, as in RGB). This is normalised to the range [0,1] by division by the maximum colour distance. The dissimilarity between a pair of points is then proportional to the colour distance and inversely proportional to the spatial distance

$$\text{diss}(p, q) = \frac{d_c(p, q)}{1 + g \cdot d_p(p, q)} \quad (4.81)$$

where  $g$  is a constant chosen by experiment (in the original  $g = 3$ , and one can experiment). When the dissimilarity is largest, the points are most salient. The operator is actually patch based (in a manner similar to the non-local means operator) and  $p$  and  $q$  are the centre co-ordinates of an  $N \times N$  patch. A pixel  $a$  is considered to be salient if the appearance of the patch centred at the pixel  $a$  is distinctive with respect to all other image patches. The colour distance is then evaluated for two patches centred at  $a$  and  $b$

$$d_{\text{patch}}(a, b) = \sum_{x(b) \in N} \sum_{y(b) \in N} \text{diss}(a, b) \quad (4.82)$$

Rather than use all the distances between patches, the operator uses the best  $K$  dissimilarities (the ones with the largest values). Again, the value for  $K$  can be chosen by experiment. The saliency at point  $a$  is determined by averaging the best  $K$  dissimilarities over patches  $b$  as

$$S(a) = 1 - e^{-\frac{\sum_{k=1}^K d_{\text{patch}}(a, b)}{K}} \quad (4.83)$$

and has a low value when the point is similar to the other image points and is closer to 1.0 when it is dissimilar. This can be viewed as an operation at a chosen scale. If the image is resampled, or the operator arranged to process points at different intervals (or the image is iteratively smoothed) then the saliency at scale  $s$  is

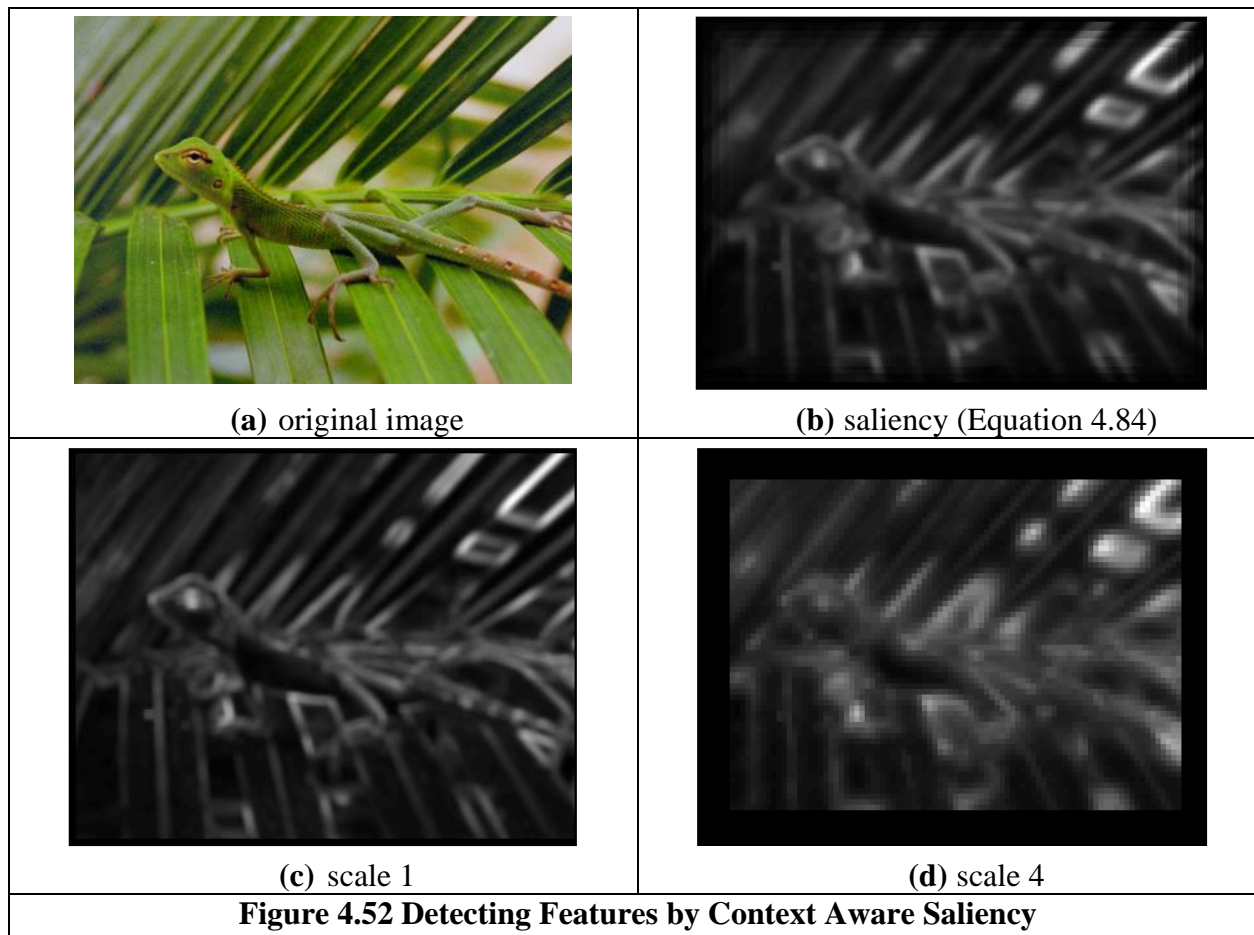
$$S(a, s) = 1 - e^{-\frac{\sum_{k=1}^K d_{\text{patch}}(a, b, s)}{K}} \quad (4.84)$$

The overall saliency is then the average of the saliency over the number of scales  $NS$

$$S(a) = \frac{1}{NS} \sum_{s=1}^{NS} S(a, s) \quad s \in 1, NS \quad (4.85)$$

The process is illustrated in Fig 4.52 for settings  $K = 64$ ,  $g = 3$  and  $NS = 4$ . Fig 4.52 shows the original image (in printed versions a colour version of the original is to be found later) and the

saliency at scales 1 and 4 (Equation 4.84) and the overall saliency over the four scales (Equation 4.85). Note that the eye features prominently as do some of the leaves (but not all). These are to be expected. There is also a region near the lizard’s front left foot, which on closer inspection stands out, but is less apparent to human vision when looking at the whole image: one does not see a combination of colour and spatial difference, as exemplified by the context aware saliency operator.



This is neither the whole story nor the full one. Given the difficulty of exact implementation of any technique, one should consult Goferman’s original version [Goferman12], and later any variants. Potential differences include the nature of the spatial operator and the normalisation procedures. There is also a question of windowing and the version here used a basis similar to that of the non-local means smoothing operator (Section 3.5.3) and here the patch size was  $7 \times 7$  and the search region  $13 \times 13$  (a limitation of larger search size is that the inverse relationship with distance makes a large search infeasible as well as slow). One can of course analyse the result of processing the same images as in the original version, and the differences are more in the details. The original version was extended to include the immediate context and that has not been included here.

#### 4.4.3.3 Other Saliency Operators

Methods that use local contrast, similar to context aware saliency, tend to produce higher saliency values near edge points instead of uniformly highlighting salient objects as can be observed in Figure 4.52(b). The newer *histogram contrast* method [Cheng15] measures saliency by evaluating pixel-wise saliency values using *colour* separation from all other image pixels. The saliency of a pixel is thus defined using its colour (again the CIE Lab) difference from all other pixels in the image

$$S(p) = \sum_{q \in \text{Image}} d_c(p, q) \quad (4.86)$$

Expanding this as a list of differences for  $N$  image points

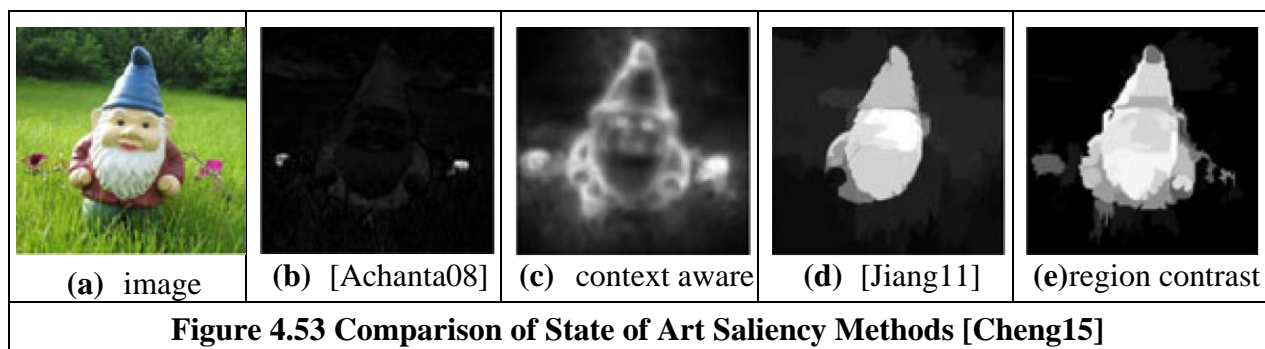
$$S(p) = d_c(p, I_1) + d_c(p, I_2) + \dots + d_c(p, I_N) \quad (4.87)$$

By this, points with the same colour have the same saliency value as we have only considered colour not space. By grouping the terms with the same colour value

$$S(p) = S(c_l) = \sum_{i=1}^{NC} f_i d_c(c_l, c_i) \quad (4.88)$$

where  $c_l$  is the colour of point  $p$ ,  $NC$  is the number of colours and  $f_i$  is the probability of pixel colour  $c_i$ . By comparing each point with every other point, Equation 4.86, the computation is  $O(N^2)$  whereas by Equation 4.88 the computational complexity is  $O(N + NC^2)$ . So to speed up the operator, one needs to compress the colour space, to reduce  $NC$ . It is not the time to dwell on colour here, and this is achieved by colour quantisation and smoothing. The extension to include spatial awareness, called the region contrast operator, employs colour region segmentation and contrast with distance regions attracting low weight (as in contrast aware saliency).

As shown in Figure 4.53 there is a variety of operators for saliency. Given the image of the gnome in Figure 4.53(a) there are two objects behind it and the gnome has eyes and a hat. One operator detects the objects (b), one the eyes, the edges and the objects (c), just the gnome's head and beard (in sections) (d), and the objects and the (sectioned) gnome (e). The latter operator is the region contrast refined version of the histogram contrast method described in Equation 4.88. Clearly, saliency can be suck and see, as are many computer vision operators, and there is a rich selection of diverse techniques by which a version can be obtained. After all, what indeed is saliency? None of the approaches includes shape, implicitly and intentionally, and shape comes in the next Chapter.



There has been a series of benchmarking studies for the performance (accuracy and speed) of the saliency operators. The most recent [Borji15] noted the continuing progress improving both factors, and that operators designed specifically for saliency outperformed those which were derived in other areas. There has been interest in using *deep learning* which aims for better performance and at speed and this has motivated *fast analysis of saliency* [Wang18]. Another study uses autoencoders [Li 18] (see Chapter 12) and also introduces a dataset for video saliency analysis. Perhaps this is the way to achieve saliency at video rate. A, perhaps unwelcome, recent development is to refer to saliency as Salient Object Detection and thus by the acronym SOD. Though it will allow native speakers of English to vent their frustration, we avoided the temptation to use that acronym here.





## 4.5 Further Reading

This Chapter has covered the main ways to extract low-level feature information. There has been an enormous amount of work in low-level features since this is a primary feature extraction task in Computer Vision and Image Processing. In some cases this can prove sufficient for understanding the image. Often though, the function of low-level feature extraction is to provide information for later higher level analysis. This can be achieved in a variety of ways, with advantages and disadvantages and quickly or at a lower speed (or requiring a faster processor/ more memory!). The range of techniques presented here has certainly proved sufficient for the majority of applications. There are other, more minor techniques, but the main approaches to boundary, corner, feature and motion extraction have proved sufficiently robust and with requisite performance that they shall endure for some time. Given depth and range, the further reading for each low level operation is to be found at the end of each section. We shall collect the most recent work on Deep Learning in Chapter 12 and shall find that the techniques from this Chapter are important in the new architectures for image analysis.

We now move on to using this information at a higher level. This means collecting the information so as to find shapes and objects, the next stage in understanding the image's content.

## 4.6 Chapter 4 References

- [Alahi12] Alahi, A., Ortiz, R. and Vandergheynst, P., Freak: Fast retina keypoint. *Proc. CVPR*, pp. 510-517, 2012
- [Achanta08] Achanta, R., Estrada, F., Wils, P. and Süsstrunk, S., Salient region detection and segmentation. In *Int. Conf. on Computer Vision Systems*, pp. 66-75, 2008
- [Adelson85] Adelson, E. H., and Bergen, J. R., Spatiotemporal Energy Models for the Perception of Motion, *Journal of the Optical Society of America*, **A2**(2), pp 284-299, 1985
- [Agrawal08] Agrawal, M., Konolige, K. and Blas, M. R., Censure: Center surround extremas for realtime feature detection and matching, *Proc. ECCV*, pp. 102-115, 2008
- [Apostol66] Apostol, T. M., *Calculus*, 2<sup>nd</sup> Ed., **1**, Xerox College Publishing, Waltham, 1966
- [Asada86] Asada, H., and Brady, M., The Curvature Primal Sketch, *IEEE Trans. on PAMI*, **8**(1), pp 2-14, 1986
- [Bailer15] Bailer, C., Taetz, B. and Stricker, D., Flow fields: Dense correspondence fields for highly accurate large displacement optical flow estimation. *Proc. CVPR*, 2015
- [Baker11] Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J. and Szeliski, R., A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, **92**(1), pp.1-31, 2011
- [Barnard87] Barnard, S. T., and Fichler, M. A., Stereo Vision, in *Encyclopedia of Artificial Intelligence*, New York: John Wiley, pp 1083-2090, 1987
- [Bay06] Bay, H., Tuytelaars, T., and Van Gool, L., SURF: Speeded Up Robust Features, *Proc. ECCV 2006*, pp 404-417, 2006
- [Bay08] Bay, H., Eas, A., Tuytelaars, T., and Van Gool, L., Speeded-Up Robust Features (SURF), *Computer Vision and Image Understanding*, **110**(3), pp 346-359, 2008

- [Bennet75] Bennet, J. R., and MacDonald, J. S., On the Measurement of Curvature in a Quantised Environment, *IEEE Trans. on Computers*, **C-24**(8), pp 803-820, 1975
- [Bergholm87] Bergholm, F., Edge Focussing, *IEEE Trans. on PAMI*, **9**(6), pp 726-741, 1987
- [Borji15] Borji, A., Cheng, M.M., Jiang, H. and Li, J., Salient object detection: A benchmark. *IEEE Trans. on IP*, **24**(12), pp.5706-5722, 2015
- [Bovik87] Bovik, A. C., Huang, T. S., and Munson, D. C., The Effect of Median Filtering on Edge Estimation and Detection, *IEEE Trans. on PAMI*, **9**(2), pp 181-194, 1987
- [Calonder10] Calonder, M., Lepetit, V., Strecha, C. and Fua, P., BRIEF: Binary robust independent elementary features. Proc. *ECCV*, pp. 778-792, 2010
- [Canny86] Canny, J., A Computational Approach to Edge Detection, *IEEE Trans. on PAMI*, **8**(6), pp 679-698, 1986
- [Cheng15] Cheng, M. M., Mitra, N. J., Huang, X., Torr, P. H. and Hu, S.M., Global contrast based salient region detection. *IEEE Trans. on PAMI*, **37**(3), pp. 569-582, 2015
- [Clark89] Clark, J. J., Authenticating Edges Produced by Zero-Crossing Algorithms, *IEEE Trans. on PAMI*, **11**(1), pp 43-57, 1989
- [Davies05] Davies, E. R., *Machine Vision: Theory, Algorithms and Practicalities*, Morgan Kaufmann (Elsevier), 3<sup>rd</sup> Edition, 2005.
- [Deriche87] Deriche, R., Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector, *International Journal of Computer Vision*, **1**, pp 167-187, 1987
- [Dhond89] Dhond, U. R. and Aggarwal J. K., Structure From Stereo- a Review, *IEEE Trans. on Systems, Man and Cybernetics*, **19**(6), pp 1489-1510, 1989
- [Dollár15] Dollár, P., and Zitnick, C. L., Fast edge detection using structured forests, *IEEE Trans. on PAMI*, **37**(8), pp 1558-1570, 2015
- [Dosovitskiy15] Dosovitskiy, A., Fischer, P., Ilg, E., Hausser, P., Hazirbas, C., Golkov, V., Van Der Smagt, P., Cremers, D. and Brox, T., FlowNet: Learning optical flow with convolutional networks. Proc. *ICCV*, pp. 2758-2766, 2015
- [Fergus03] Fergus, R., Perona, P., and Zisserman, A., Object Class Recognition by Unsupervised Scale-Invariant Learning, Proc. *CVPR 2003*, **II**, pp 264-271, 2003
- [Fleet90] Fleet, D. J. and Jepson, A. D., Computation of Component Image Velocity from Local Phase Information, *International Journal of Computer Vision*, **5**(1), pp. 77-104, 1990
- [Forshaw88] Forshaw, M. R. B., Speeding Up the Marr-Hildreth Edge Operator, *CVGIP*, **41**, pp 172-185, 1988
- [Fortun15] Fortun, D., Bouthemy, P. and Kervrann, C., Optical flow modeling and computation: a survey. *Computer Vision and Image Understanding*, **134**, pp.1-21, 2015
- [Goetz70] Goetz, A., *Introduction to Differential Geometry*, Addison-Wesley, Reading MA USA, 1970
- [Goferman12] Goferman, S., Zelnik-Manor, L. and Tal, A., Context-aware saliency detection. *IEEE Trans. on PAMI*, **34**(10), pp.1915-1926, 2012
- [Grimson85] Grimson, W. E. L., and Hildreth, E. C., Comments on "Digital Step Edges from Zero Crossings of Second Directional Derivatives", *IEEE Trans. on PAMI*, **7**(1), pp 121-127, 1985
- [Groan78] Groan, F., and Verbeek, P., Freeman-Code Probabilities of Object Boundary Quantized Contours, *Comput. Vision, Graphics, Image Processing*, **7**, pp 391-402, 1978
- [Gunn99] Gunn, S. R., On the Discrete Representation of the Laplacian of Gaussian, *Pattern Recognition*, **32**(8), pp 1463-1472, 1999

- [Haddon88] Haddon, J. F., Generalised Threshold Selection for Edge Detection, *Pattern Recognition*, **21**(3), pp 195-203, 1988
- [Harris88] Harris, C., and Stephens, M., A Combined Corner and Edge Detector, *Proc. Fourth Alvey Vision Conference*, pp 147-151, 1988
- [Haralick84] Haralick, R. M., Digital Step Edges from Zero-Crossings of Second Directional Derivatives, *IEEE Trans. on PAMI*, **6**(1), pp 58-68, 1984
- [Haralick85] Haralick, R. M., Author's Reply, *IEEE Trans. on PAMI*, **7**(1), pp 127-129, 1985
- [Heath97] Heath, M. D., Sarkar, S., Sanocki, T., and Bowyer, K. W., A Robust Visual Method of Assessing the Relative Performance of Edge Detection Algorithms, *IEEE Trans. on PAMI*, **19**(12), pp 1338-1359, 1997
- [Horn81] Horn, B. K. P., and Schunk, B. G., Determining Optical Flow, *Artificial Intelligence*, **17**, pp 185-203, 1981
- [Horn86] Horn, B. K. P., *Robot Vision*, MIT Press, Cambridge, 1986
- [Horn93] Horn, B. K. P., and Schunk, B. G., Determining Optical Flow: a Retrospective, *Artificial Intelligence*, **59**, pp 81-87, 1993
- [Huang99] Huang, P. S., Harris, C. J. and Nixon, M. S., Human Gait Recognition in Canonical Space using Temporal Templates, *IEE Proceedings Vision Image and Signal Processing*, **146**(2), pp 93-100, 1999
- [Huertas86] Huertas, A., and Medioni, G., Detection of Intensity Changes with Subpixel Accuracy using Laplacian-Gaussian Masks, *IEEE Trans. on PAMI*, **8**(1), pp 651-664, 1986
- [Ilg17] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A. and Brox, T., FlowNet 2.0: Evolution of optical flow estimation with deep networks. *Proc. CVPR*, 2017
- [Jia95] Jia, X., and Nixon, M. S., Extending the Feature Vector for Automatic Face Recognition, *IEEE Trans. on PAMI*, **17**(12), pp 1167-1176, 1995
- [Jiang11] Jiang, H., Wang, J., Yuan, Z., Liu, T., Zheng, N. and Li, S., Automatic salient object segmentation based on context and shape prior. In *Proc. BMVC*, 2011
- [Jordan92] Jordan III, J. R. and Bovik A. C., M. S., Using Chromatic Information in Dense Stereo Correspondence, *Pattern Recognition*, **25**, pp 367-383, 1992
- [Kadir01] Kadir, T., and Brady, M., Scale, Saliency and Image Description, *International J. Computer Vision*. **45**(2), pp 83-105, 2001
- [Kanade94] Kanade, T. and Okutomi, M., A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment, *IEEE Trans. on PAMI*, **16**, pp 920-932, 1994
- [Kass88] Kass, M., Witkin, A. and Terzopoulos, D., Snakes: Active Contour Models, *Int. J. Computer Vision*, **1**(4), pp 321-331, 1988
- [Ke04] Y. Ke and R. Sukthankar. PCA-sift: A more distinctive representation for local image descriptors. *Proc. CVPR 2004*, II, pp 506–513, 2004.
- [Kimia18] Kimia, B. B., Li, X., Guo, Y., and Tamrakar, A., Differential geometry in edge detection: accurate estimation of position, orientation and curvature, *IEEE Trans. on PAMI*, DOI 10.1109/TPAMI.2018.2846268, 2018
- [Kitchen82] Kitchen, L. and Rosenfeld, A., Gray-Level Corner Detection, *Pattern Recog. Lett.*, **1**(2), pp 95-102, 1982
- [Korn88] Korn, A. F., Toward a Symbolic Representation of Intensity Changes in Images, *IEEE Trans. on PAMI*, **10**(5), pp 610-625, 1988
- [Kovesi99] Kovesi, P., Image Features from Phase Congruency. *Videre: Journal of Computer Vision Research*, **1**(3), pp 1–27, 1999

- [Lawton83] Lawton, D. T., Processing Translational Motion Sequences, *Computer Vision, Graphics and Image Processing*, **22**, pp 116-144, 1983
- [LeCun15] LeCun, Y., Bengio, Y. and Hinton, G., Deep learning, *Nature*, **521**(7553), pp.437-444, 2015
- [Lee93] Lee, C. K., Haralick, M., Deguchi, K., Estimation of Curvature from Sampled Noisy Data, *ICVPR'93*, pp 536-541, 1993
- [Leutenegger11] Leutenegger, S., Chli, M. and Siegwart, R.Y., 2011, BRISK: Binary robust invariant scalable keypoints. *Proc. ICCV*, pp. 2548-2555, 2011
- [Li 18] Li, J., Xia, C. and Chen, X., A benchmark dataset and saliency-guided stacked autoencoders for video-based salient object detection. *IEEE Trans. on IP*, **27**(1), pp.349-364, 2018
- [Lindeberg94] Lindeberg, T., Scale-Space Theory: a Basic Tool for Analysing Structures at Different Scales. *Journal of Applied Statistics*, **21**(2), pp 224-270, 1994
- [Little98] Little, J. J. and Boyd, J. E., Recognizing People By Their Gait: The Shape of Motion, *Videre*, **1**(2), pp 2-32, 1998, online at <http://mitpress.mit.edu/e-journals/VIDE/001/v12.html>
- [Lomeli16] Lomeli-R, J., and Nixon, M. S., An extension to the brightness clustering transform and locally contrasting keypoints, *Machine Vis. and Apps.*, **27**(8) pp. 1187-1196, 2016
- [Lowe99] Lowe, D. G., Object Recognition from Local Scale-Invariant Features, *Proc. ICCV*, pp 1150-1157, 1999
- [Lowe04] Lowe, D. G., Distinctive Image Features from Scale-Invariant Key points, *International J. of Computer Vision*, **60**(2), pp 91-110, 2004
- [Lucas81] Lucas, B., and Kanade, T., An Iterative Image Registration Technique with an Application to Stereo Vision, *Proc DARPA Image Understanding Workshop*, pp 121-130, 1981
- [Marr80] Marr, D. C., and Hildreth, E., Theory of Edge Detection, *Proc. Royal Society of London*, **B207**, pp 187-217, 1980
- [Marr82] Marr, D., *Vision*, W. H. Freeman and Co., N.Y. USA, 1982
- [Mikolajczyk05] Mikolajczyk, K., and Schmid, C., A Performance Evaluation of Local Descriptors, *IEEE Trans. on PAMI*, **27**(10), pp 1615- 1630, 2005
- [Mokhtarian86] Mokhtarian F., and Mackworth, A. K., Scale-Space Description and Recognition of Planar Curves and Two-Dimensional Shapes, *IEEE Trans. on PAMI*, **8**(1), pp 34-43, 1986
- [Mokhtarian03] Mokhtarian F., and Bober, M., *Curvature Scale Space Representation: Theory, Applications and MPEG-7 Standardization*, Kluwer Academic Publishers, 2003
- [Morrone87] Morrone, M. C., and Owens, R. A., Feature Detection from Local Energy, *Pattern Recognition Letters*, **6**, pp 303–313, 1987.
- [Morrone88] Morrone M. C., and Burr, D. C., Feature Detection in Human Vision: a Phase-Dependent Energy Model, *Proceedings of the Royal Society of London, Series B, Biological Sciences*, **235**(1280), pp 221–245, 1988
- [Mukherjee15] Mukherjee, D., Wu, Q. J. and Wang, G., A comparative experimental study of image feature detectors and descriptors. *Machine Vis. and Apps.*, **26**(4), pp.443-466, 2015
- [Mulet-Parada00] Mulet-Parada, M., and Noble, J. A., 2D+T Acoustic Boundary Detection in Echocardiography, *Medical Image Analysis*, **4**, pp 21–30, 2000
- [Myerscough04] Myerscough, P. J., and Nixon, M. S. Temporal Phase Congruency. *Proc. IEEE Southwest Symposium on Image Analysis and Interpretation SSIAI '04*, pp 76-79, 2004
- [Nagel87] Nagel, H. H., On the Estimation of Optical Flow: Relations between Different Approaches and some New Results, *Artificial Intelligence*, **33**, pp 299-324, 1987

- [Oliva06] Oliva, A. and Torralba, A., Building the gist of a scene: The role of global image features in recognition. *Progress in Brain Research*, **155**, pp.23-36, 2006
- [Otterloo91] van Otterloo, P. J., *A Contour-Oriented Approach to Shape Analysis*, Prentice Hall International (UK) Ltd., Hemel Hempstead UK, 1991
- [Petrou91] Petrou, M., and Kittler, J., Optimal Edge Detectors for Ramp Edges, *IEEE Trans. on PAMI*, **13**(5), pp 483-491, 1991
- [Petrou94] Petrou, M., The Differentiating Filter Approach to Edge Detection, *Advances in Electronics and Electron Physics*, **88**, pp 297-345, 1994
- [Prewitt66] Prewitt, J. M. S. and Mendelsohn, M. L., The Analysis of Cell Images, *Ann. N. Y. Acad. Sci.*, **128**, pp 1035-1053, 1966
- [Revaud15] Revaud, J., Weinzaepfel, P., Harchaoui, Z. and Schmid, C., Epicflow: Edge-preserving interpolation of correspondences for optical flow. *Proc. CVPR*, 2015
- [Roberts65] Roberts, L. G., Machine Perception of Three-Dimensional Solids, *Optical and Electro-Optical Information Processing*, MIT Press, pp 159-197, 1965
- [Rosin96] Rosin, P. L., Augmenting Corner Descriptors, *Graphical Models and Image Processing*, **58**(3), pp 286-294, 1996
- [Rosten06] Rosten, E., and Drummond, T., Machine learning for high-speed corner detection, *Proc. ECCV*, pp. 430-443, 2006
- [Rublee11] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., ORB: An efficient alternative to SIFT or SURF, *Proc. ICCV*, pp. 2564-2571, 2011
- [Smith97] Smith, S. M. and Brady, J. M., SUSAN - A New Approach to Low Level image Processing. *Int. Journal of Computer Vision*, **23**(1), pp 45-78, May 1997
- [Sobel70] Sobel, I. E., *Camera Models and Machine Perception*, PhD Thesis, Stanford Univ., 1970
- [Spacek86] Spacek, L. A., Edge Detection and Motion Detection, *Image and Vision Computing*, **4**(1), pp 43-56, 1986
- [Sun14] Sun, D., Roth, S. and Black, M. J., A quantitative analysis of current practices in optical flow estimation and the principles behind them. *Int. J. Computer Vision*, **106**(2), pp.115-137, 2014
- [Sun18] Sun, Y., Hare, J. S. and Nixon, M. S., Detecting heel strikes for gait analysis through acceleration flow. *IET Computer Vision*, **12**(5), pp 686-692, 2018
- [Torre86] Torre, V., and Poggio, T. A., On Edge Detection, *IEEE Trans. on PAMI*, **8**(2), pp 147-163, 1986
- [Tuytelaars07] Tuytelaars, T., and Mikolajczyk, K., Local Invariant Feature Detectors: A Survey, *Foundations and Trends in Computer Graphics and Vision*, **3**(3), pp 177–280, 2007
- [Ulupinar90] Ulupinar, F., and Medioni, G., Refining Edges Detected by a LoG Operator, *CVGIP*, **51**, pp 275-298, 1990
- [Venkatesh89] Venkatesh, S., and Owens, R. A. An Energy Feature Detection Scheme. *Proc. of an International Conference on Image Processing*, pp 553–557, Singapore, 1989
- [Venkatesh95] Venkatesh, S, and Rosin, P. L., Dynamic Threshold Determination by Local and Global Edge Evaluation, *Graphical Models and Image Processing*, **57**(2), pp146-160, 1995
- [Vliet89] Vliet, L. J., and Young, I. T., A Nonlinear Laplacian Operator as Edge Detector in Noisy Images, *CVGIP*, **45**, pp 167-195, 1989
- [Wang18] Wang, W., Shen, J. and Shao, L., Video salient object detection via fully convolutional networks. *IEEE Trans. on IP*, **27**(1), pp.38-49, 2018

- [Weinzaepfel13] Weinzaepfel, P., Revaud, J., Harchaoui, Z. and Schmid, C., DeepFlow: Large displacement optical flow with deep matching. *Proc. ICCV*, pp. 1385-1392, 2013
- [Yitzhaky03] Yitzhaky, Y., and Peli, E., A Method for Objective Edge Detection Evaluation and Detector Parameter Selection, *IEEE Trans. on TPAMI*, **25**(8), pp 1027-1033, 2003
- [Zabir94] Zabir, R., and Woodfill, J., Non\_parametric Local Transforms for Computing Visual Correspondence, *Proc. ECCV*, pp 151-158,1994
- [Zheng04] Zheng, Y., Nixon, M. S., and Allen R., Automatic Segmentation of Lumbar Vertebrae in Digital Videofluoroscopic Imaging, *IEEE Trans. on Medical Imaging*, **23**(1), pp 45-52 2004
- [Zheng18] Zheng, L., Yang, Y. and Tian, Q., SIFT meets CNN: A decade survey of instance retrieval, *IEEE Trans. on PAMI*, **40**(5), pp.1224-1244, 2018

- 7×7 Sobel, 134
- acceleration, 193
- analysis, 129
- aperture, 187
- autocorrelation*, 167
- averaging, 131
- backward*, 164
- band-pass, 150
- Binary Robust Invariant Scalable Keypoints, 177
- BRISK*, 177
- Canny, 138, 153, 155
- CBIR*, 175
- Content Based Image Retrieval*, 175
- context aware saliency, 179
- contrast, 127
- Convolutional Neural Networks
  - keypoint, 177
- corner, 160
- curvature, 160, 161, 163, 167
- curvature function*, 161
- DeepFlow, 192
- derivative of Gaussian*, 138
- edge, 125, 127, 130, 131, 135
- Epicflow, 192
- Evaluating Optical Flow**, 195
- Evaluation Optical Flow**, 195
- FAST, 176
- Fast Retina Keypoint*, 177
- fast saliency*, 182
- Features from Accelerated Segment Test*, 176
- first, 127, 153
- First order*, 145
- forward*, 164
- Fourier, 136
- FREAK*, 177
- Gaussian, 138, 139, 147
- Gaussian (LoG) operator*, 147
- general, 133
- GIST*, 177
- high-pass, 136
- histogram contrast saliency, 181
- horizontal, 127, 187
- hysteresis, 139, 141, 142
- inverse Fourier transform, 155
- Laplacian, 146, 147
- line, 164
- Locally Contrasting Keypoints*, 177
- LOCKY*, 177
- low-level, 125
- low-pass, 136, 150
- Marr-Hildreth, 147, 153
- Mexican, 147
- Middlebury, 194
- motion, 183
- moving object*, 183
- non-maximum, 138, 139, 140
- optical*, 183, 186
- optical flow evaluation, 194
- optimal, 138
- ORB, 177
- Oriented FAST and Rotated BRIEF*, 177
- performance, 178
- Petrou, 152
- Phase Congruency, 154
- planar, 161
- Prewitt, 131, 153
- Roberts, 129
- saliency*, 179
- saliency deep learning, 182
- second, 145
- second order*, 145
- smoothness, 188
- Sobel, 132, 135, 139, 153
- Spacek, 153
- STAR*, 177
- template, 132, 136
- uniform, 128, 143
- vertical, 127, 187
- wavelet*, 157
- zero-crossing, 145, 149





## 4 Low-Level Feature Extraction (including Edge Detection) 125

4.1 Overview.....	125
4.2 Edge Detection.....	127
4.2.1 First Order Edge Detection Operators .....	127
4.2.1.1 <i>Basic Operators</i> .....	127
4.2.1.2 <i>Analysis of the Basic Operators</i> .....	129
4.2.1.3 <i>Prewitt Edge Detection Operator</i> .....	131
4.2.1.4 <i>Sobel Edge Detection Operator</i> .....	132
4.2.1.5 <i>The Canny Edge Detector</i> .....	138
4.2.2 Second Order Edge Detection Operators .....	145
4.2.2.1 <i>Motivation</i> .....	145
4.2.2.2 <i>Basic Operators: The Laplacian</i> .....	146
4.2.2.3 <i>The Marr-Hildreth Operator</i> .....	147
4.2.3 Other Edge Detection Operators .....	151
4.2.4 Comparison of Edge Detection Operators .....	153
4.2.5 Further Reading on Edge Detection.....	153
4.3 Phase Congruency.....	154
4.4 Localised Feature Extraction .....	159
4.4.1 Detecting Image Curvature (Corner Extraction).....	160
4.4.1.1 <i>Definition of Curvature</i> .....	160
4.4.1.2 <i>Computing Differences in Edge Direction</i> .....	161
4.4.1.3 <i>Measuring Curvature by Changes in Intensity (Differentiation)</i> .....	163
4.4.1.4 <i>Moravec and Harris Detectors</i> .....	167
4.4.1.5 <i>Further Reading on Curvature</i> .....	170
4.4.2 Feature Point Detection; Region/Patch Analysis .....	171
4.4.2.1 <i>Scale Invariant Feature Transform (SIFT)</i> .....	171
4.4.2.2 <i>Speeded Up Robust Features (SURF)</i> .....	174
4.4.2.3 <i>FAST, ORB, FREAK, LOCKY and other keypoint detectors</i> .....	175
4.4.2.4 <i>Other Techniques and Performance Issues</i> .....	178
4.4.3 Saliency.....	179
4.4.3.1 <i>Basic Saliency</i> .....	179
4.4.3.2 <i>Context Aware Saliency</i> .....	179
4.4.3.3 <i>Other Saliency Operators</i> .....	181
4.5 Describing Image Motion .....	183
4.5.1 Area-based approach.....	<b>Error! Bookmark not defined.</b>
4.5.2 Differential approach .....	<b>Error! Bookmark not defined.</b>
4.5.3 Recent Developments: Deepflow, Epicflow and extensions ..	<b>Error! Bookmark not defined.</b>
4.5.4 Analysis of Optical Flow .....	<b>Error! Bookmark not defined.</b>

4.6 Further Reading .....	184
4.7 Chapter 4 References .....	184

## 5 High-Level Feature Extraction: Fixed Shape Matching

### 5.1 Overview

High-level *feature extraction* concerns finding shapes and objects in computer images. To be able to recognise human faces automatically, for example, one approach is to extract the component features. This requires extraction of, say, the eyes, the ears and the nose, which are the major face features. To find them, we can use their shape: the white part of the eyes is ellipsoidal; the mouth can appear as two lines, as do the eyebrows. Alternatively, we can view them as objects and use the low-level features to define collections of points, which define the eyes, nose and mouth, or even the whole face. This feature extraction process can be viewed as similar to the way we perceive the world: many books for babies describe basic geometric shapes such as triangles, circles and squares. More complex pictures can be decomposed into a structure of simple shapes. In many applications, analysis can be guided by the way the shapes are arranged. For the example of face image analysis, we expect to find the eyes above (and either side of) the nose and we expect to find the mouth below the nose.

In feature extraction, we generally seek *invariance* properties so that the extraction result does not vary according to chosen (or specified) conditions. This implies finding objects, whatever their position, their orientation or their size. That is, techniques should find shapes reliably and robustly whatever the value of any parameter that can control the appearance of a shape. As a basic invariant, we seek immunity to changes in the illumination level: we seek to find a shape whether it is light or dark. In principle, as long as there is contrast between a shape and its background, the shape can be said to exist, and can then be detected. (Clearly, any computer vision technique will fail in extreme lighting conditions; you cannot see anything when it is completely dark.) Following illumination, the next most important parameter is position: we seek to find a shape wherever it appears. This is usually called *position-*, *location-* or *translation-invariance*. Then, we often seek to find a shape irrespective of its rotation (assuming that the object or the camera has an unknown orientation): this is usually called *rotation-* or *orientation-invariance*. Then, we might seek to determine the object at whatever size it appears, which might be due to physical change, or to how close the object has been placed to the camera. This requires *size-* or *scale-invariance*. These are the main invariance properties we shall seek from our shape extraction techniques. However, nature (as usual) tends to roll balls under our feet: there is always noise in images. Also since we are concerned with shapes, note that there might be more than one in the image. If one is on top of the other it will occlude, or hide, the other, so not all the shape of one object will be visible.

But before we can develop image analysis techniques, we need techniques to extract the shapes and objects. Extraction is more complex than detection, since extraction implies that we have a description of a shape, such as its position and size, whereas detection of a shape merely implies knowledge of its existence within an image. This Chapter concerns shapes that are fixed in shape (such as a segment of bone in a medical image); the following Chapter concerns shapes, which can deform (like the shape of a walking person).

The techniques presented in this chapter are outlined in Table 5.1. We first consider whether we can detect objects by thresholding. This is only likely to provide a solution when illumination and lighting can be controlled, so we then consider two main approaches: one is to extract constituent parts; the other is to extract constituent shapes. We can actually collect and describe low-level features described earlier. In this, wavelets can provide object descriptions, as can Scale-Invariant Feature Transform SIFT and distributions of low-level features. In this way we represent objects as a collection of interest points, rather than using shape analysis. Conversely, we can investigate the

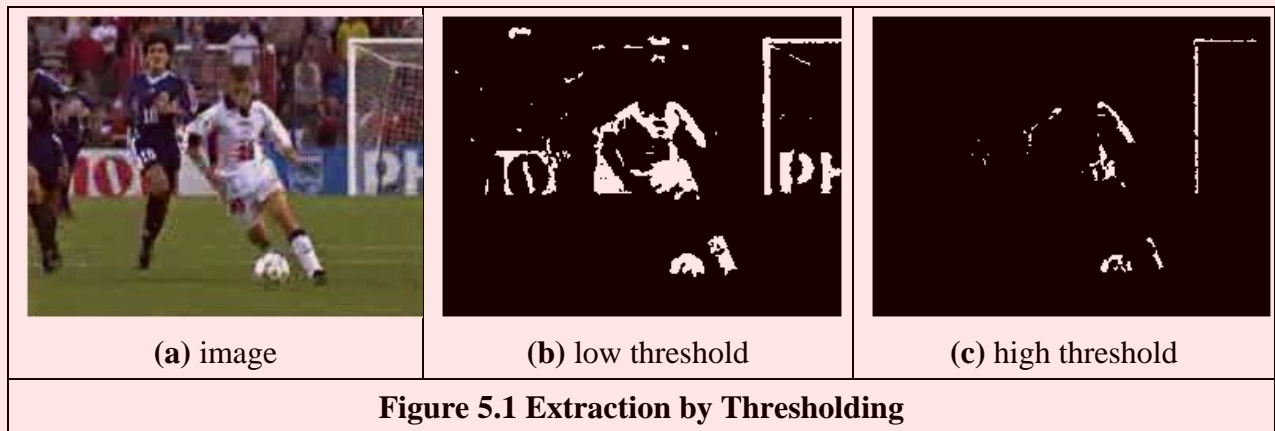
use of shape: template matching is a model-based approach in which the shape is extracted by searching for the best correlation between a known model and the pixels in an image. There are alternative ways in to compute the correlation between the template and the image. Correlation can be implemented by considering the image or frequency domains and the template can be defined by considering intensity values or a binary shape. The Hough transform defines an efficient implementation of template matching for binary templates. This technique is capable of extracting simple shapes such as lines and quadratic forms as well as arbitrary shapes. In any case, the complexity of the implementation can be reduced by considering invariant features of the shapes

Main topic	Sub topics	Main points
Pixel operations	How we detect features at a pixel level. What are the limitations and advantages of this approach. Need for shape information.	<i>Thresholding. Differencing.</i>
Template matching	Shape extraction by matching. Advantages and disadvantages. Need for efficient implementation.	<i>Template Matching. Direct and Fourier implementations. Noise and occlusion.</i>
Low level features	Collecting low level features for object extraction and their descriptive bases. Frequency-based and parts-based approaches. Detecting distributions of measures.	<i>Wavelets and Haar wavelets. SIFT, SURF, BRIEF and ORB descriptions and Histogram of Oriented Gradients.</i>
Hough transform	Feature extraction by matching. Hough transforms for conic sections. Hough transform for arbitrary shapes. Invariant formulations. Advantages in speed and efficacy.	Feature extraction by evidence gathering. <i>Hough transforms</i> for lines, circles and ellipses. <i>Generalised and Invariant Hough transforms.</i>
<b>Table 5.1 Overview of Chapter 5</b>		

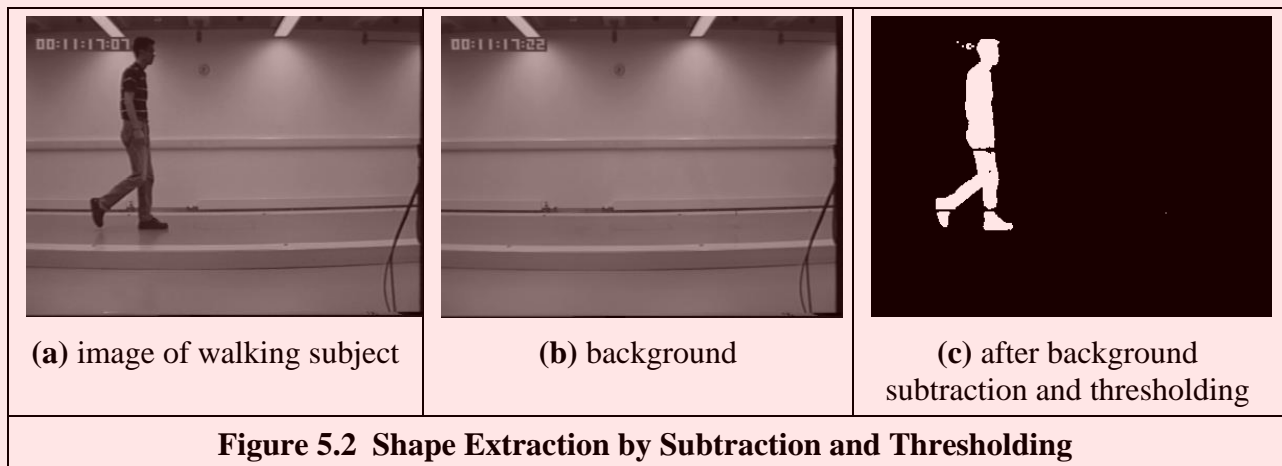
## 5.2 Thresholding and Subtraction

Thresholding is a simple shape extraction technique, as illustrated in Section 3.3.4, where the images could be viewed as the result of trying to separate the eye from the background. If it can be assumed that the shape to be extracted is defined by its brightness, then thresholding an image at that brightness level should find the shape. Thresholding is clearly sensitive to change in illumination: if the image illumination changes then so will the perceived brightness of the target shape. Unless the threshold level can be arranged to adapt to the change in brightness level, any thresholding technique will fail. Its attraction is simplicity: thresholding does not require much computational effort. If the illumination level changes in a linear fashion, using histogram equalisation will result in an image that does not vary. Unfortunately, the result of histogram equalisation is sensitive to noise, shadows and variant illumination: noise can affect the resulting image quite dramatically and this will again render a thresholding technique useless. Let us illustrate this by considering Figure 5.1 and let us consider trying to find either the ball, or the player, or both in Figure 5.1(a). Superficially, these are the brightest objects so one value of the threshold, Figure 5.1(b), finds the player's top, shorts and socks, and the ball – but it also finds the text in the advertising and the goalmouth. When we increase the threshold, Figure 5.1(c), we lose

the advertising, parts of the player, but still find the goalmouth. Clearly we need to include more knowledge, or to process the image more.



Thresholding after intensity normalisation (Section 3.3.2) is less sensitive to noise, since the noise is stretched with the original image, and cannot affect the stretching process by much. It is however still sensitive to shadows and variant illumination. Again, it can only find application where the illumination can be carefully controlled. This requirement is germane to any application that uses basic thresholding. If the overall illumination level cannot be controlled, it is possible to threshold edge magnitude data since this is insensitive to overall brightness level, by virtue of the implicit differencing process. However, edge data is rarely continuous and there can be gaps in the detected perimeter of a shape. Another major difficulty, which applies to thresholding the brightness data as well, is that there are often more shapes than one. If the shapes are on top of each other, one occludes the other and the shapes need to be separated.



An alternative approach is to subtract an image from a known background before thresholding. This assumes that the background is known precisely, otherwise many more details than just the target feature will appear in the resulting image; clearly the subtraction will be unfeasible if there is noise on either image, and especially on both. In this approach, there is no implicit shape description, but if the thresholding process is sufficient, it is simple to estimate basic shape parameters, such as position.

The subtraction approach is illustrated in Figure 5.2. Here, we seek to separate or extract a walking subject from their background. When we subtract the background of Figure 5.2(b) from the image itself, we obtain most of the subject with some extra background just behind the subject's

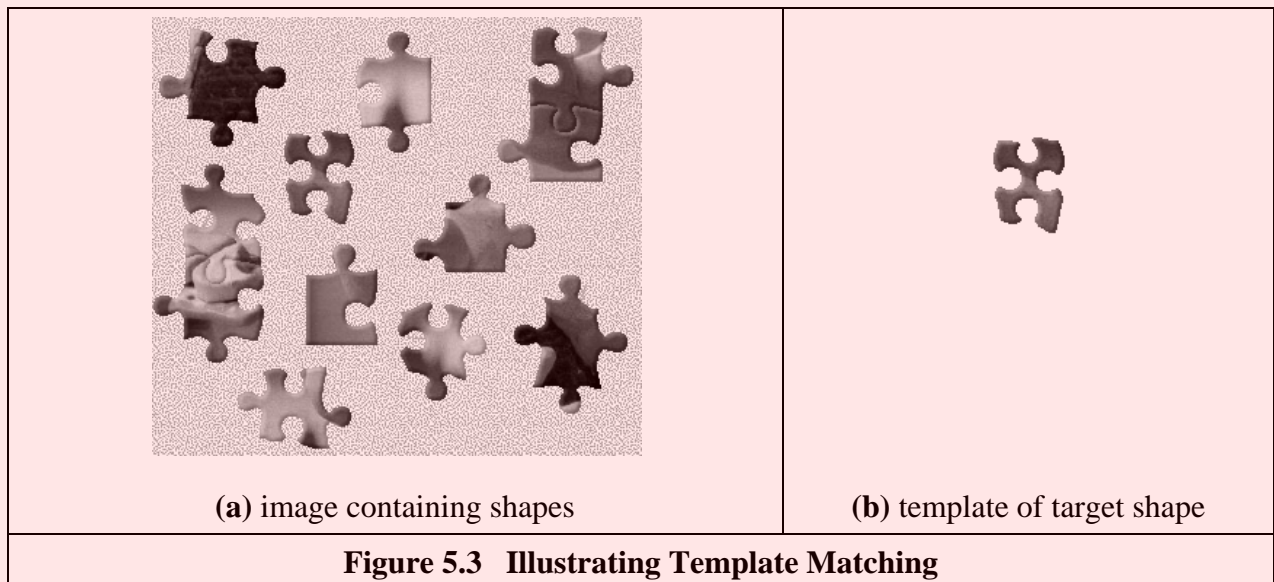
head (this is due to the effect of the moving subject on lighting). Also, removing the background removes some of the subject: the horizontal bars in the background have been removed from the subject by the subtraction process. These aspects are highlighted in the thresholded image, Figure 5.2(c). It is not a particularly poor way of separating the subject from the background (we have the subject but we have chopped through his midriff) but it is not especially good either. So it does provide an estimate of the object, but an estimate that is only likely to be reliable when the lighting is highly controlled. (A more detailed study of separation of moving objects from their static background, including estimation of the background itself, is to be found in Chapter 9.)

Even though thresholding and subtraction are attractive (because of simplicity and hence their speed), the performance of both techniques is sensitive to partial shape data, to noise, variation in illumination and to occlusion of the target shape by other objects. Accordingly, many approaches to image interpretation use higher level information in shape extraction, namely how the pixels are connected. This can resolve these factors.

## 5.3 Template Matching

### 5.3.1 Definition

*Template matching* is conceptually a simple process. We need to match a template to an image, where the template is a sub-image that contains the shape we are trying to find. Accordingly, we centre the template on an image point and count how many points in the template matched those in the image. The procedure is repeated for the entire image and the point which led to the best match, the maximum count, is deemed to be the point where the shape (given by the template) lies within the image.



Consider that we want to find the template of Figure 5.3(b) in the image of Figure 5.3(a). The template is first positioned at the origin and then matched with the image to give a count that reflects how well the template matched that part of the image at that position. The count of matching pixels is increased by one for each point where the brightness of the template matches the brightness of the image. This is similar to the process of template convolution, illustrated earlier in Figure 3.11. The difference here is that points in the image are matched with those in the template, and the sum is of the number of matching points as opposed to the weighted sum of image data. The best match

which gives the maximum number of matching points is when the template is placed at the position where the target shape is matched to itself. Obviously, this process can be generalised to find, for example, templates of different size or orientation. In these cases, we have to try all the templates (at expected rotation and size) to determine the best match.

Formally, template matching can be defined as a method of parameter estimation. The parameters define the position (and pose) of the template. We can define a template as a discrete function  $\mathbf{T}_{x,y}$ . This function takes values in a window. That is, the co-ordinates of the points  $(x, y) \in \mathbf{W}$ . For example, for a  $2 \times 2$  template we have the set of points  $\mathbf{W} = \{(0,0), (0,1), (1,0), (1,1)\}$ . In general, a template can be defined by considering any property from an image. The example in Figure 5.4 shows templates defined by grey values, binary values and image edges. Other templates can also define colours or multispectral data.

Let us consider that each pixel in the image  $\mathbf{I}_{x,y}$  is corrupted by additive Gaussian noise. The noise has a mean value of zero and the (unknown) standard deviation is  $\sigma$ . Thus, the probability that a point in the template placed at co-ordinates  $(i, j)$  matches the corresponding pixel at position  $(x, y) \in \mathbf{W}$  is given by the normal distribution

$$p_{i,j}(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \quad (5.1)$$

Since the noise affecting each pixel is independent, then the probability that the template is at position  $(i, j)$  is the combined probability of each pixel that the template covers. That is,

$$L_{i,j} = \prod_{(x,y) \in \mathbf{W}} p_{i,j}(x, y) \quad (5.2)$$

By substitution of Equation 5.1, we have that

$$L_{i,j} = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n e^{-\frac{1}{2} \sum_{(x,y) \in \mathbf{W}} \left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \quad (5.3)$$

where  $n$  is the number of pixels in the template. This function is called the *likelihood* function. Generally, it is expressed in logarithmic form to simplify the analysis. Notice that the logarithm scales the function, but it does not change the position of the maximum. Thus, by taking the logarithm the likelihood function is redefined as

$$\ln(L_{i,j}) = n \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2} \sum_{(x,y) \in \mathbf{W}} \left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2 \quad (5.4)$$

In *maximum likelihood estimation*, we have to choose the parameter that maximises the likelihood function. That is, the positions that minimise the rate of change of the objective function

$$\frac{\partial \ln(L_{i,j})}{\partial i} = 0 \quad \text{and} \quad \frac{\partial \ln(L_{i,j})}{\partial j} = 0 \quad (5.5)$$

That is,

$$\begin{aligned} \sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial i} &= 0 \\ \sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial j} &= 0 \end{aligned} \quad (5.6)$$

We can observe that these equations are also the solution of the minimisation problem given by

$$\min e = \sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y})^2 \quad (5.7)$$

That is, maximum likelihood estimation is equivalent to choosing the template position that minimises the squared error (the squared values of the differences between the template points and the corresponding image points). The position where the template best matches the image is the estimated position of the template within the image. Thus, if you measure the match using the squared error criterion, then you will be choosing the *maximum likelihood* solution. This implies that the result achieved by template matching is optimal for images corrupted by Gaussian noise. (Note that the *Central Limit Theorem* suggests that noise experienced practically can be assumed to be Gaussian distributed, though many images appear to contradict this assumption.) Of course you can use other error criteria such as the absolute difference rather than the squared difference or, if you feel more adventurous, you might consider robust measures such as M-estimators.

We can derive alternative forms of the squared error criterion by considering that Equation (5.7) can be written as

$$\min e = \sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j}^2 - 2\mathbf{I}_{x+i,y+j} \mathbf{T}_{x,y} + \mathbf{T}_{x,y}^2) \quad (5.8)$$

The last term does not depend on the template position  $(i, j)$ . As such, it is constant and cannot be minimised. Thus, the optimum in this equation can be obtained by minimising

$$\min e = \sum_{(x,y) \in \mathbf{W}} \left( \mathbf{I}_{x+i,y+j}^2 - 2 \sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j} \mathbf{T}_{x,y} \right) \quad (5.9)$$

If the first term

$$\sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j}^2 \quad (5.10)$$

is approximately constant, then the remaining term gives a measure of the similarity between the image and the template. That is, we can maximise the cross-correlation between the template and the image. Thus, the best position can be computed by

$$\max e = \sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j} \mathbf{T}_{x,y} \quad (5.11)$$

However, the squared term in Equation (5.10) can vary with position, so the match defined by Equation (5.11) can be poor. Additionally, the range of the cross-correlation is dependent on the size of the template and it is non-invariant to changes in image lighting conditions. Thus, in an implementation it is more convenient to use either Equation (5.7) or Equation (5.9) (in spite of being computationally more demanding than the cross-correlation in Equation 5.11). Alternatively, cross-correlation can be normalised as follows. We can rewrite Equation (5.9) as

$$\min e = 1 - 2 \frac{\sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j} \mathbf{T}_{x,y}}{\sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j}^2} \quad (5.12)$$

Here the first term is constant and thus, the optimum value can be obtained by

$$\max e = \frac{\sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j} \mathbf{T}_{x,y}}{\sum_{(x,y) \in \mathbf{W}} \mathbf{I}_{x+i,y+j}^2} \quad (5.13)$$

In general, it is convenient to normalise the grey level of each image window under the template. That is,



$$\max_e e = \frac{\sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j})(\mathbf{T}_{x,y} - \bar{\mathbf{T}})}{\sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j})^2} \tag{5.14}$$

where  $\bar{\mathbf{I}}_{i,j}$  is the mean of the pixels  $\mathbf{I}_{x+i,y+j}$  for points within the window (i.e.  $(x,y) \in \mathbf{W}$ ) and  $\bar{\mathbf{T}}$  is the mean of the pixels of the template. An alternative form to Equation (5.14) is given by normalising the cross-correlation. This does not change the position of the optimum and gives an interpretation as the normalisation of the cross-correlation vector. That is, the cross-correlation is divided by its modulus. Thus,

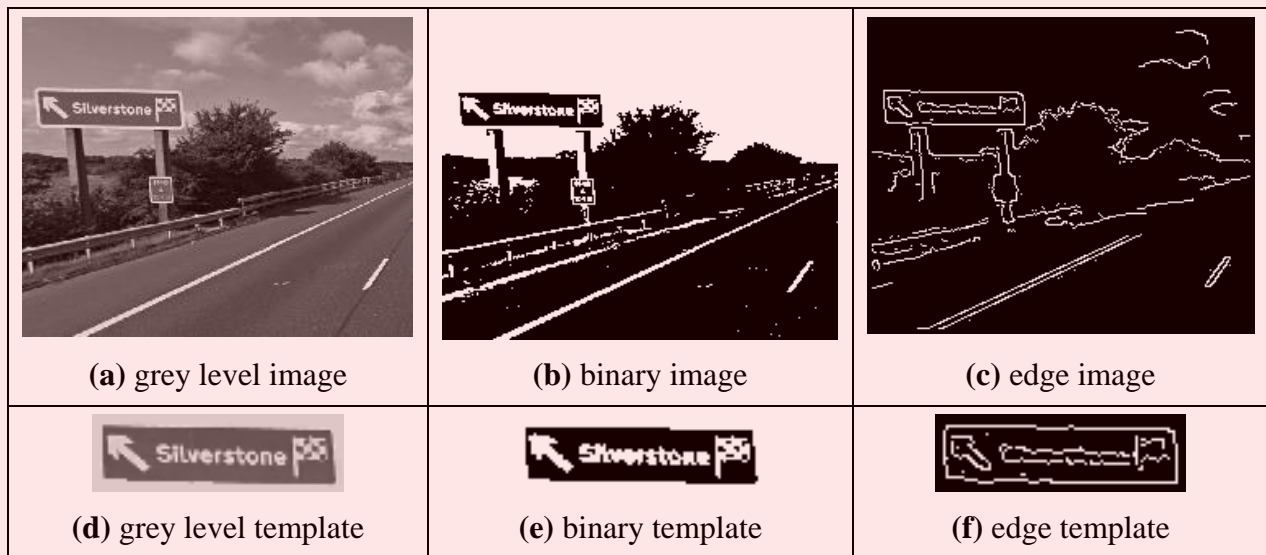
$$\max_e e = \frac{\sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j})(\mathbf{T}_{x,y} - \bar{\mathbf{T}})}{\sqrt{\sum_{(x,y) \in \mathbf{W}} (\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j})^2 (\mathbf{T}_{x,y} - \bar{\mathbf{T}})^2}} \tag{5.15}$$

However, this equation has a similar computational complexity to the original formulation in Equation (5.7).

A particular implementation of template matching is when the image and the template are binary. In this case, the binary image can represent regions in the image or it can contain the edges. These two cases are illustrated in the example in Figure 5.4. The advantage of using binary images is that the amount of computation can be reduced. That is, each term in Equation (5.7) will take only two values: it will be one when  $\mathbf{I}_{x+i,y+j} = \mathbf{T}_{x,y}$ , and zero otherwise. Thus, Equation (5.7) can be implemented as

$$\max_e e = \sum_{(x,y) \in \mathbf{W}} \overline{\mathbf{I}_{x+i,y+j} \oplus \mathbf{T}_{x,y}} \tag{5.16}$$

where the symbol  $\overline{\oplus}$  denotes the exclusive NOR operator. This equation can be easily implemented and requires significantly less resource than the original matching function.



**Figure 5.4 Examples of Grey Level, Binary and Edge Template Matching**

Template matching develops an *accumulator space* that stores the match of the template to the image at different locations; this corresponds to an implementation of Equation (5.7). It is called an accumulator, since the match is accumulated during application. Essentially, the accumulator is a two dimensional array that holds the difference between the template and the image at different positions. The position in the image gives the same position of match in the accumulator.

Alternatively, Equation (5.11) suggests that the peaks in the accumulator resulting from template correlation give the location of the template in an image: the co-ordinates of the point of best match. Accordingly, template correlation and template matching can be viewed as similar processes. The location of a template can be determined by either process. The binary implementation of template matching, Equation (5.16), usually is concerned with thresholded edge data. This equation will be reconsidered in the definition of the Hough transform, the topic of Section 5.5.

Code 5.1 shows the implementation of template matching. The code uses an accumulator array, `accumulator` and the position of the template is given by its centre. The accumulator elements are incremented according to Equation (5.7) and the match for each position is stored in the array. The implementation inverts the value of the difference in Equation (5.7), so the best match can be shown as a peak in the accumulator. As such, after computing all the matches, the maximum element in the array defines the position where most pixels in the template matched those in the image. It is possible to implement a version of template matching without the accumulator array, by only storing the location of the minimum. This will give the same result and it requires less storage. However, this implementation will provide a result that cannot support later image interpretation and that might require knowledge of more than just the best match.

```
function image = template_match(image,template)
%get image dimensions
[rows,cols]=size(image);
%get template dimensions
[trows,tcols]=size(template);
%half of template rows is
tr=floor(trows/2);
%half of template cols is
tc=floor(tcols/2);

%set an output as black
accum(1:rows,1:cols)=0;

%then convolve the template
for x = tc+1:cols-tc %address all columns except border
    for y = tr+1:rows-tr %address all rows except border
        sum=0; %initialise the sum
        for iwin=1:tcols %address all points in the template
            for jwin=1:trows
                sum=sum+image(y+jwin-tr-1,x+iwin-tc-1)*...
                    template(jwin,iwin);
            end
        end
        accum(y,x)=sum;
    end
end

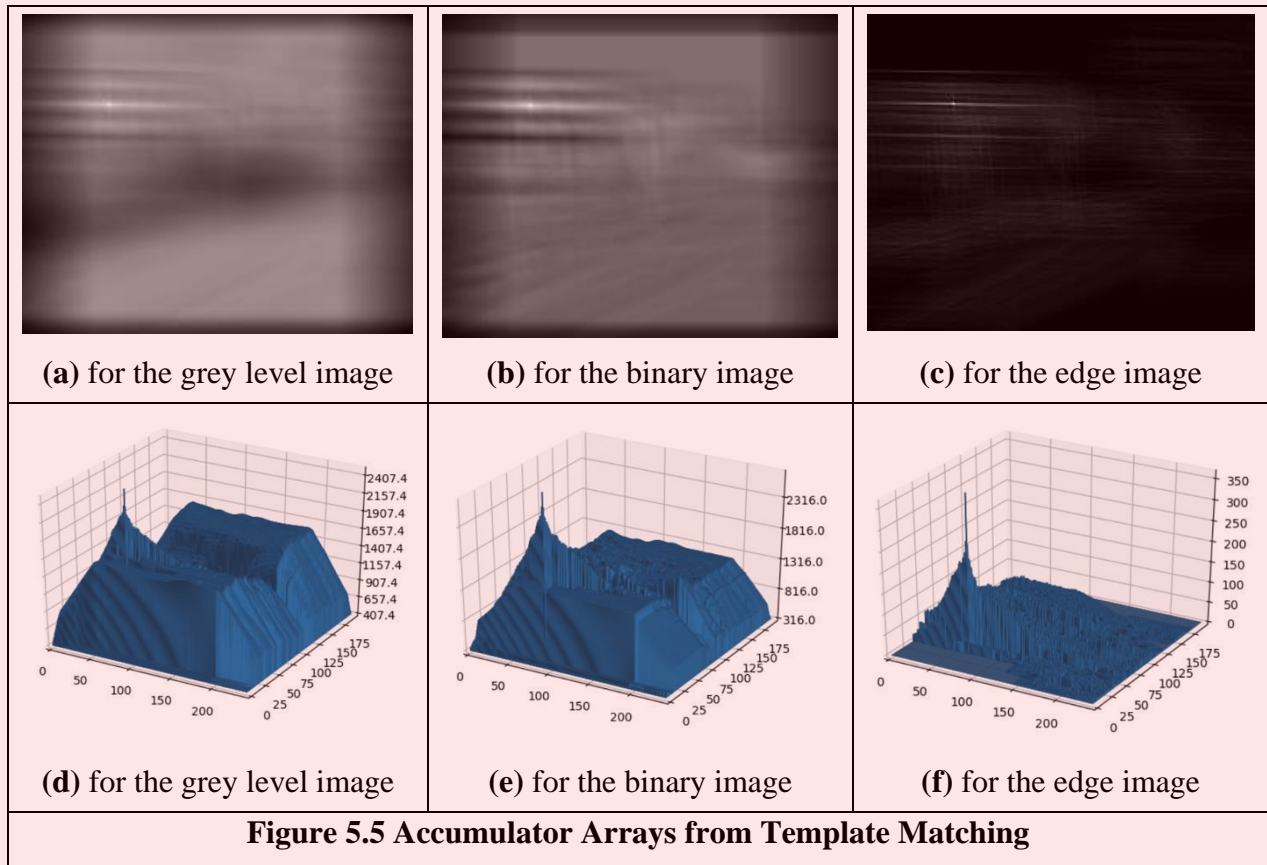
%find the maximum
biggest_vote=max(max(accum));
```

**Code 5.1 Implementing Template Matching**

The results of applying the template matching procedure are illustrated in Figure 5.5. This figure shows the accumulators as images and as histograms. The height of the histogram corresponds to the value in the accumulator. This example shows the accumulator arrays for matching the images shown in Figure 5.4 with their respective templates. The white points in each image are at the co-ordinates of the origin of the position where the template best matched the image (the maxima). Note that there is a border where the template has not been matched to the image data. At these border points, the template extended beyond the image data, so no matching has been performed. This is the same border as experienced with template convolution, Section 3.4.1. We can observe

that a clearer maximum is obtained, Figure 5.5(c), from the edge images in Figure 5.4. This is because for grey level and binary images, there is some match when the template is not exactly in the best position. In the case of edges, the count of matching pixels is less.

Most applications require further degrees of freedom such as rotation (orientation), scale (size), or perspective deformations. Rotation can be handled by rotating the template, or by using polar co-ordinates; scale invariance can be achieved using templates of differing size. Having more parameters of interest implies that the accumulator space becomes larger; its dimensions increase by one for each extra parameter of interest. Position-invariant template matching, as considered here, implies a 2D parameter space, whereas the extension to scale and position invariant template matching requires a 3D parameter space.

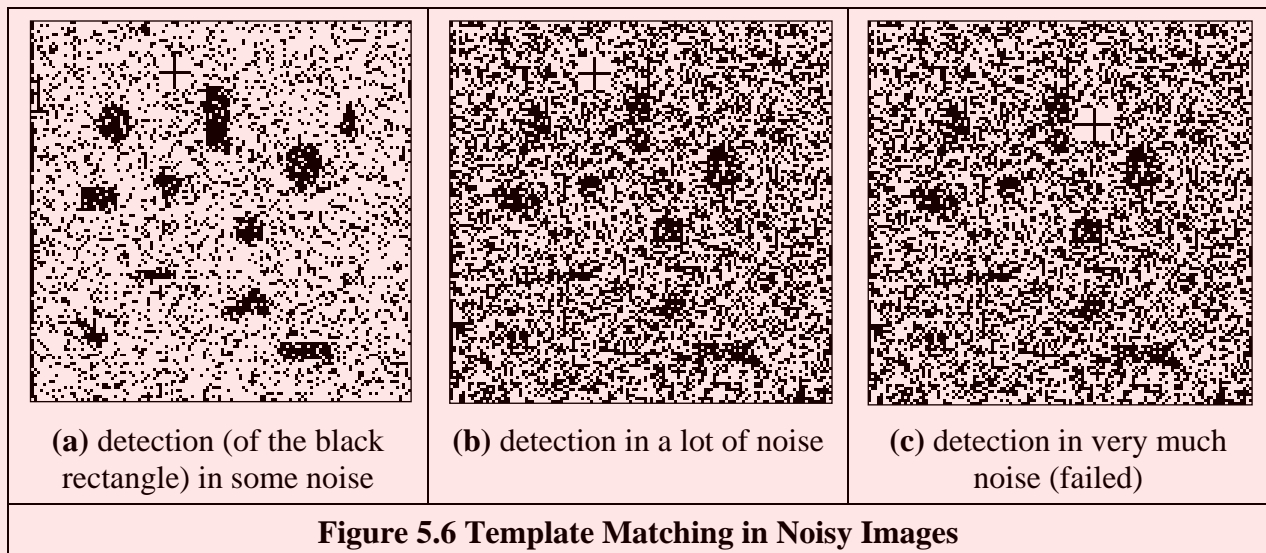


The computational cost of template matching is large. If the template is square and of size  $m \times m$  and is matched to an image of size  $N \times N$ , since the  $m^2$  pixels are matched at all image points (except for the border) the computational cost is  $O(N^2m^2)$ . This is the cost for position invariant template matching. Any further parameters of interest increase the computational cost in proportion to the number of values of the extra parameters. This is clearly a large penalty and so a direct digital implementation of template matching is slow. Accordingly, this guarantees interest in techniques that can deliver the same result, but faster, such as using a Fourier implementation based on fast transform calculus.

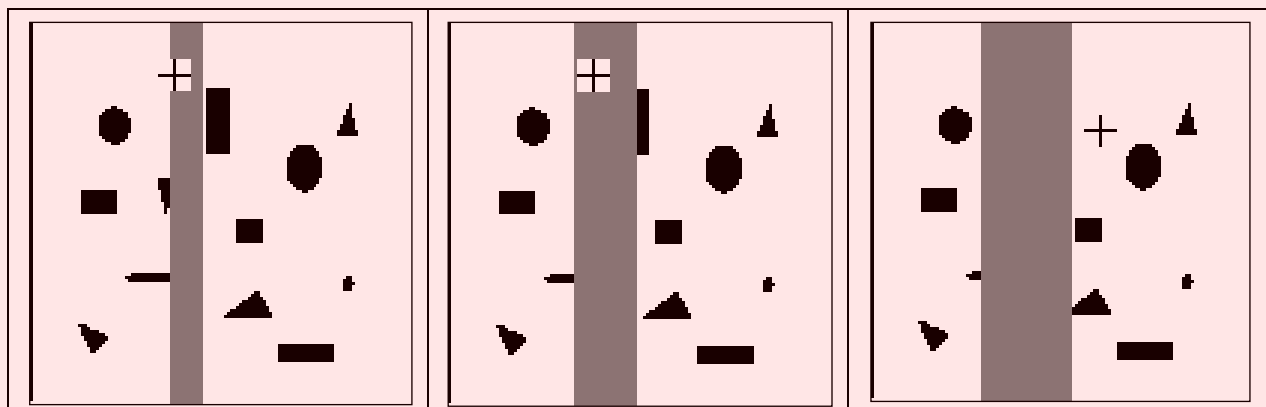
The main advantages of template matching are its insensitivity to *noise* and *occlusion*. Noise can occur in any image, on any signal – just like on a telephone line. In digital photographs, the noise might appear low, but in computer vision it is made worse by edge detection by virtue of the differencing (differentiation) processes. Likewise, shapes can easily be occluded or hidden: a person can walk behind a lamp post or illumination can also cause occlusion. The averaging

inherent in template matching reduces the susceptibility to noise; the maximisation process reduces susceptibility to occlusion.

These advantages are shown in Figure 5.6 which illustrates detection in the presence of increasing noise. Here, we will use template matching to locate the region containing the vertical rectangle near the top of the image (so we are matching a binary template of a black template on a white background to the binary image). The lowest noise level is in Figure 5.6(a) and the highest is in Figure 5.6(c); the position of the origin of the detected rectangle is shown as a black cross in a white square. The position of the origin of the region containing the rectangle is detected correctly in Figure 5.6(a) and Figure 5.6(b) but incorrectly in the noisiest image, Figure 5.6(c). Clearly, template matching can handle quite high noise corruption. (Admittedly this is somewhat artificial: the noise would usually be filtered out by one of the techniques described in Chapter 3, but we are illustrating basic properties here.) The ability to handle noise is shown by correct determination of the position of the target shape, until the noise becomes too much and there are more points due to noise than there are due to the shape itself. When this occurs, the votes resulting from the noise exceed those occurring from the shape, and so the maximum is not found where the shape exists.



Occlusion is shown by placing a grey bar across the image; in Figure 5.7(a) the bar does not occlude (or hide) the target rectangle whereas in Figure 5.7(c) the rectangle is completely obscured. As with performance in the presence of noise, detection of the shape fails when the votes occurring from the shape exceed those from the rest of the image (the non-shape points), and the cross indicating the position of the origin of the region containing the rectangle is drawn in completely the wrong place. This is what happens when the rectangle is completely obscured in Figure 5.7(c).



(a) detection (of the black rectangle) in no occlusion	(b) detection in some occlusion	(c) detection in complete occlusion (failed)
<b>Figure 5.7 Template Matching in Occluded Images</b>		

So it can operate well, with practical advantage. We can include edge detection to concentrate on a shape's borders. Its main problem is still speed: a direct implementation is slow, especially when handling shapes that are rotated or scaled (and there are other implementation difficulties too). Recalling that from Section 3.4.2 that template convolution can be speeded up by using the Fourier transform, let us see if that can be used here too.

### 5.3.2 Fourier Transform Implementation

We can implement template matching via the Fourier transform by using the duality between *convolution* and multiplication, which was discussed earlier in Section 3.4.4. This duality establishes that a multiplication in the space domain corresponds to a convolution in the frequency domain and vice versa. This can be exploited for faster computation by using the frequency domain, given the Fast Fourier Transform algorithm. Thus, in order to find a shape we can compute the cross-correlation as a multiplication in the frequency domain. However, the matching process in Equation (5.11) is actually *correlation* (Section 2.3), not convolution. Thus, we need to express the correlation in terms of a convolution. This can be done as follows. First, we can rewrite the correlation (denoted by  $\otimes$ ) in Equation (5.11) as

$$\mathbf{I} \otimes \mathbf{T} = \sum_{(x,y) \in W} \mathbf{I}_{x',y'} \mathbf{T}_{x'-i,y'-j} \quad (5.17)$$

where  $x' = x + i$  and  $y' = y + j$ . Convolution (denoted by  $*$ ) is defined as

$$\mathbf{I} * \mathbf{T} = \sum_{(x,y) \in W} \mathbf{I}_{x',y'} \mathbf{T}_{i-x',j-y'} \quad (5.18)$$

Thus, in order to implement template matching in the frequency domain, we need to express Equation (5.17) in terms of Equation (5.18). This can be achieved by considering that the correlation

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \sum_{(x,y) \in W} \mathbf{I}_{x',y'} \mathbf{T}'_{i-x',j-y'} \quad (5.19)$$

where

$$\mathbf{T}' = \mathbf{T}_{-x,-y} \quad (5.20)$$

That is, correlation is equivalent to convolution when the template is changed according to Equation (5.20). This equation reverses the co-ordinate axes and it corresponds to a horizontal and a vertical flip.

In the frequency domain, convolution corresponds to multiplication. As such, we have that Equation (5.19) can be implemented by

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \mathfrak{F}^{-1}(\mathfrak{F}(\mathbf{I}) \times \mathfrak{F}(\mathbf{T}')) \quad (5.21)$$

where  $\mathfrak{F}$  denotes Fourier transformation as in Chapter 2 (and calculated by the FFT) and  $\mathfrak{F}^{-1}$  denotes the inverse FFT. Note that the multiplication operator actually operates point-by-point, so each point is the product of the pixels at the same position in each image (in Matlab the operation is  $\cdot$ ). This is computationally faster than its direct implementation, given the speed advantage of the FFT. There are two ways to implement this equation. In the first approach, we can compute  $\mathbf{T}'$

by flipping the template and then computing its Fourier transform  $\mathfrak{Z}(\mathbf{T}')$ . In the second approach, we compute the transform of  $\mathfrak{Z}(\mathbf{T})$  and then we compute its complex conjugate. That is,

$$\mathfrak{Z}(\mathbf{T}') = [\mathfrak{Z}(\mathbf{T})]^* \quad (5.22)$$

where  $[ ]^*$  denotes the complex conjugate of the transform data (yes, we agree it's an unfortunate symbol clash with convolution, but they are both standard symbols). So conjugation of the transform of the template implies that the product of the two transforms leads to correlation. (Since this product is point-by-point the two images/ matrices need to be of the same size.) That is,

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \mathfrak{Z}^{-1}(\mathfrak{Z}(\mathbf{I}) \times [\mathfrak{Z}(\mathbf{T})]^*) \quad (5.23)$$

For both implementations, Equations (5.21) and (5.23) will evaluate the match and, more quickly for large templates than by direct implementation of template matching (as per Section 3.4). Note that one assumption is that the transforms are of the same size, even though the template's shape is usually much smaller than the image. There is actually a selection of approaches; a simple solution is to include extra zero values (*zero-padding*) to make the image of the template the same size as the image.

Code 5.2 illustrates the implementation of template matching by convolution in the image domain. In this implementation the input image is padded to include the size of the template. Figure 5.8(a) shows the padding pixels in black. Padding is not necessary when computing convolution in the image domain, but it is included so the results can be compared with the computation of the convolution in the frequency domain in Code 5.3. The main loop in Code 5.2 computes the error criterion in Equation (5.8). The code separates the computation of the correlation terms and those terms which are squared. The term which is squared is computed separately so it can be used later in Code 5.3. An example of the result of the template matching by convolution in the image domain is shown in Figure 5.8(d). The white point in the image defines the maximum value and indicates the position of the template. The peak is more evident in Figure 5.8(g) where the accumulator is shown as a histogram.

```
# Pad input
inputPad = createImageF(widthPad, heightPad)
for x,y in itertools.product(range(0, width), range(0, height)):
    inputPad[y,x] = inputImage[y,x]

# Compute correlation in image domain sum of square differences
squaredTerm = createImageF(widthPad, heightPad)
corrImage = createImageF(widthPad, heightPad)
for x,y in itertools.product(range(0, widthPad), range(0, heightPad)):
    for w,h in itertools.product(range(-widthTemplate+1,1), \
                                range(-heightTemplate+1,1)):
        p, q = x+w, y+h
        if p >=0 and q >= 0 and p < width and q < height:
            squaredTerm[y,x] += inputPad[q,p] * inputPad[q,p]
            corrImage[y,x] += 2.0 * templatePad[h+heightTemplate,w+widthTemplate] \
                               * inputPad[q,p]
```

### Code 5.2 Implementing Convolution in the Image Domain

Code 5.3 illustrates the computation of template matching by convolution in the frequency domain. Similar to Code 5.2, the input image is padded. In this case, the template is also rotated and padded according to Equation (5.19). Figure 5.8(b) shows the rotated and padded template. The implementation in Code 5.3 computes the Fourier coefficients of the image and of the template and it stores them in the arrays `imageCoeff` and `templateCoeff`. These coefficients are then used to compute the summation in Equation (5.19). The summation is stored in the `resultCoeff`

array. Notice that the summation computes the convolution since the template has been rotated. The power spectrum of the convolution is shown in Figure 5.8(c). This corresponds to the frequency representation of the convolution. The function `reconstruction` obtains the image representation of the convolution by reconstructing the image from the `resultCoeff` array. Figures 5.8(e) show the result of the Frequency domain convolution. Notice that the best match is located, but the peak is less evident than the peak in Figure 5.8(d). In order to make the convolution equivalent to the sum of squared differences, the code adds the quadratic term computed in Code 5.2. The result of the summation is shown in Figures 5.8(f) and 5.8(i). When the quadratic term is included, then the results are equivalent to the results in Figures 5.8(a) and 5.8(b). The only difference is how the convolution is computed (i.e., in the image or in the frequency domain). In practice the computation in the frequency domain is more expensive and the summation in the image domain is simpler to implement and it can be easily organized in a parallel way, thus the image domain convolutions are generally preferred in applications. Notice that the results contain several small local maxima (in white). This can be explained by the fact that the shape can partially match several patterns in the image. Also we can see that, in contrast to the computation in the image domain, the implementation in the frequency domain does not have any border. This is due to the fact that Fourier theory assumes picture *replication* to infinity.

Should we seek scale invariance, to find the position of a template irrespective of its size, then we need to formulate a set of templates that range in size between the maximum and minimum expected variation. Each of the templates of differing size is then matched by frequency domain multiplication. The maximum frequency domain value, for all sizes of template, indicates the position of the template and, naturally, gives a value for its size. This can of course be a rather lengthy procedure when the template ranges considerably in size.

```
# Pad and invert template
templatePadFlip = createImageF(widthPad, heightPad)
for x,y in itertools.product(range(0, widthTemplate), range(0, heightTemplate)):
    templatePadFlip[y,x] = templateImage[heightTemplate-y-1, widthTemplate-x-1]

# Compute Fourier coefficients
imageCoeff, maxFrequencyW, maxFrequencyH = computeCoefficients(inputPad)
templateCoeff, _, _ = computeCoefficients(templatePadFlip)

# Frequency domain multiplication defines convolution in space domain
resultCoeff = createImageF(1 + 2 * maxFrequencyW, 1 + 2 * maxFrequencyH, 2)
for kw,kh in itertools.product(range(-maxFrequencyW, maxFrequencyW + 1), \
                               range(-maxFrequencyH, maxFrequencyH + 1)):
    w = kw + maxFrequencyW
    h = kh + maxFrequencyH
    resultCoeff[h,w][0] = (imageCoeff[h,w][0] * templateCoeff[h,w][0] - \
                          imageCoeff[h,w][1] * templateCoeff[h,w][1])
    resultCoeff[h,w][1] = (imageCoeff[h,w][1] * templateCoeff[h,w][0] + \
                          imageCoeff[h,w][0] * templateCoeff[h,w][1])

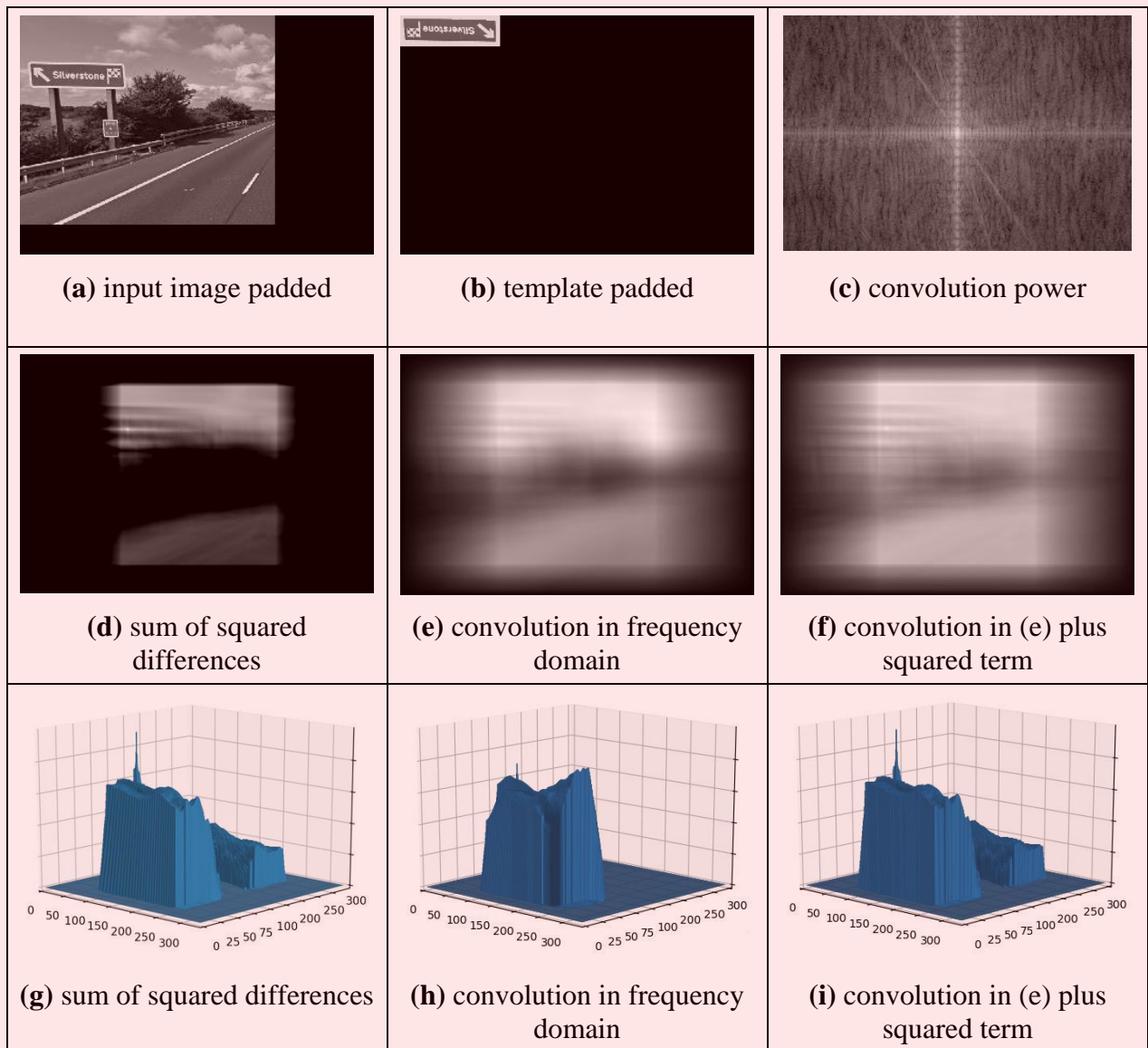
# Inverse Fourier transform
reconstructedResult = reconstruction(resultCoeff)

# Add square term to define an operator equivalent to SSD
for x,y in itertools.product(range(0, widthPad), range(0, heightPad)):
    reconstructedResult[y,x] = -squaredTerm[y,x] + 2.0 * reconstructedResult[y,x]
```

### Code 5.3 Implementing Convolution by the Frequency Domain

There are several further difficulties in using the transform domain for template matching in discrete images. If we seek rotation invariance, then an image can be expressed in terms of its polar co-ordinates. Discretisation gives further difficulty since the points in a rotated discrete shape can map imperfectly to the original shape. This problem is better manifest when an image is scaled in

size to become larger. In such a case, the spacing between points will increase in the enlarged image. The difficulty is how to allocate values for pixels in the enlarged image that are not defined in the enlargement process. There are several interpolation approaches, but it can often appear prudent to reformulate the original approach. Further difficulties can include the influence of the image borders: Fourier theory assumes that an image replicates spatially to infinity. Such difficulty can be reduced by using window operators, such as the Hamming or the Hanning windows. These difficulties do not obtain for optical Fourier transforms and so using the Fourier transform for position-invariant template matching is often confined to optical implementations.



**Figure 5.8 Template Matching by Fourier Transformation**

### 5.3.3 Discussion of Template Matching

The advantages associated with template matching are mainly theoretical since it can be very difficult to develop a template matching technique that operates satisfactorily. The results presented here have been for position invariance only. This can cause difficulty if invariance to rotation and scale is also required. This is because the template is stored as a discrete set of points. When these



are rotated, gaps can appear due to the discrete nature of the co-ordinate system. If the template is increased in size then again there will be missing points in the scaled-up version. Again, there is a frequency domain version that can handle variation in size, since scale invariant template matching can be achieved using the *Mellin transform* [Bracewell86]. This avoids using many templates to accommodate the variation in size by evaluating the scale-invariant match in a single pass. The Mellin transform essentially scales the spatial co-ordinates of the image using an exponential function. A point is then moved to a position given by a logarithmic function of its original co-ordinates. The transform of the scaled image is then multiplied by the transform of the template. The maximum again indicates the best match between the transform and the image. This can be considered to be equivalent to a change of variable. The logarithmic mapping ensures that scaling (multiplication) becomes addition. By the logarithmic mapping, the problem of scale invariance becomes a problem of finding the position of a match.

The Mellin transform only provides scale-invariant matching. For scale and position invariance, the Mellin transform is combined with the Fourier transform, to give the *Fourier-Mellin* transform. The Fourier-Mellin transform has many disadvantages in a digital implementation, due to the problems in spatial resolution though there are approaches to reduce these problems [Altmann84], as well as the difficulties with discrete images experienced in Fourier transform approaches.

Again, the Mellin transform appears to be much better suited to an optical implementation [Casasent77], where continuous functions are available, rather than to discrete image analysis. A further difficulty with the Mellin transform is that its result is independent of the form factor of the template. Accordingly, a rectangle and a square appear to be the same to this transform. This implies a loss of information since the form factor can indicate that an object has been imaged from an oblique angle. There has been some interest in *log-polar mappings* for image analysis (e.g. [Zokai05]).

So there are innate difficulties with template matching whether it is implemented directly, or by transform operations. For these reasons, and because many shape extraction techniques require more than just edge or brightness data, direct digital implementations of feature extraction are usually preferred. This is perhaps also influenced by the speed advantage that one popular technique can confer over template matching. This is the Hough transform, which is covered in Section 5.5. Before that, we shall consider techniques which consider object extraction by collections of low level features. These can avoid the computational requirements of template matching by treating shapes as collections of features.

## 5.4 Feature Extraction by Low Level Features

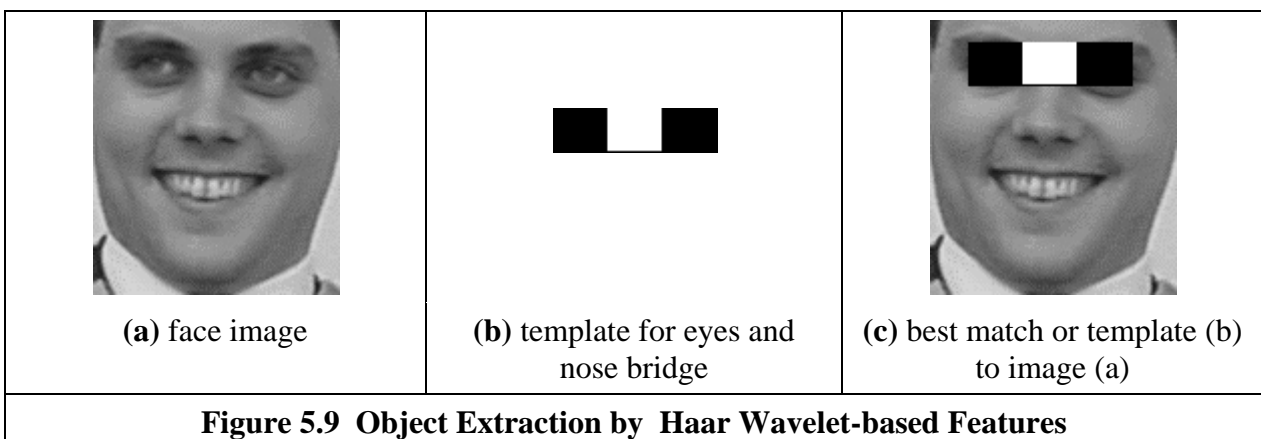
There have been many approaches to feature extraction which combine a variety of features. It is possible to characterise objects by measures that we have already developed, by low level features local features (such edges and corners), and by global features (such as colour). Later we shall find these can be grouped to give structure or shape (in this Chapter and the next), and appearance (called texture, Chapter 8). The drivers for the earlier approaches which combine low level features are the need to be able to search databases for particular images. This is known as image retrieval (Section 4.4.2) and in content-based retrieval that uses techniques from image processing and computer vision there are approaches that combine a selection of features [Smeulders00]. Alternative search strategies include using text or sketches and these are not of interest in the domain of this book. One approach is to develop features which include and target human descriptions and use techniques from machine intelligence [Datta 08], which also implies understanding of semantics (how people describe images) as compared with the results of automated image analysis.

There is also interest in recognising objects, and hence images, by collecting descriptors for local features [Mikolajczyk05]. These can find application not just in image retrieval but also in stereo computer vision, navigating robots by computer vision and when stitching together multiple images to build a much larger panorama image. Much this material relates to whole applications and therefore can rely not just on collecting local features, on shape, on texture and on classification. In these respects in this Chapter we shall provide coverage of some of the basic ways to combine low-level feature descriptions. Essentially, these approaches show how techniques that have already been covered can be combined in such a way as to achieve a description by which an object can be recognised. The approaches tend to rely on the use of machine learning approaches to determine the relevant data (to filter it so as to understand its structure) so the approaches are described in basis only here and the classification approaches are described later, in Chapter 12.

### 5.4.1 Appearance-Based Approaches

#### 5.4.1.1 Object Detection by Templates

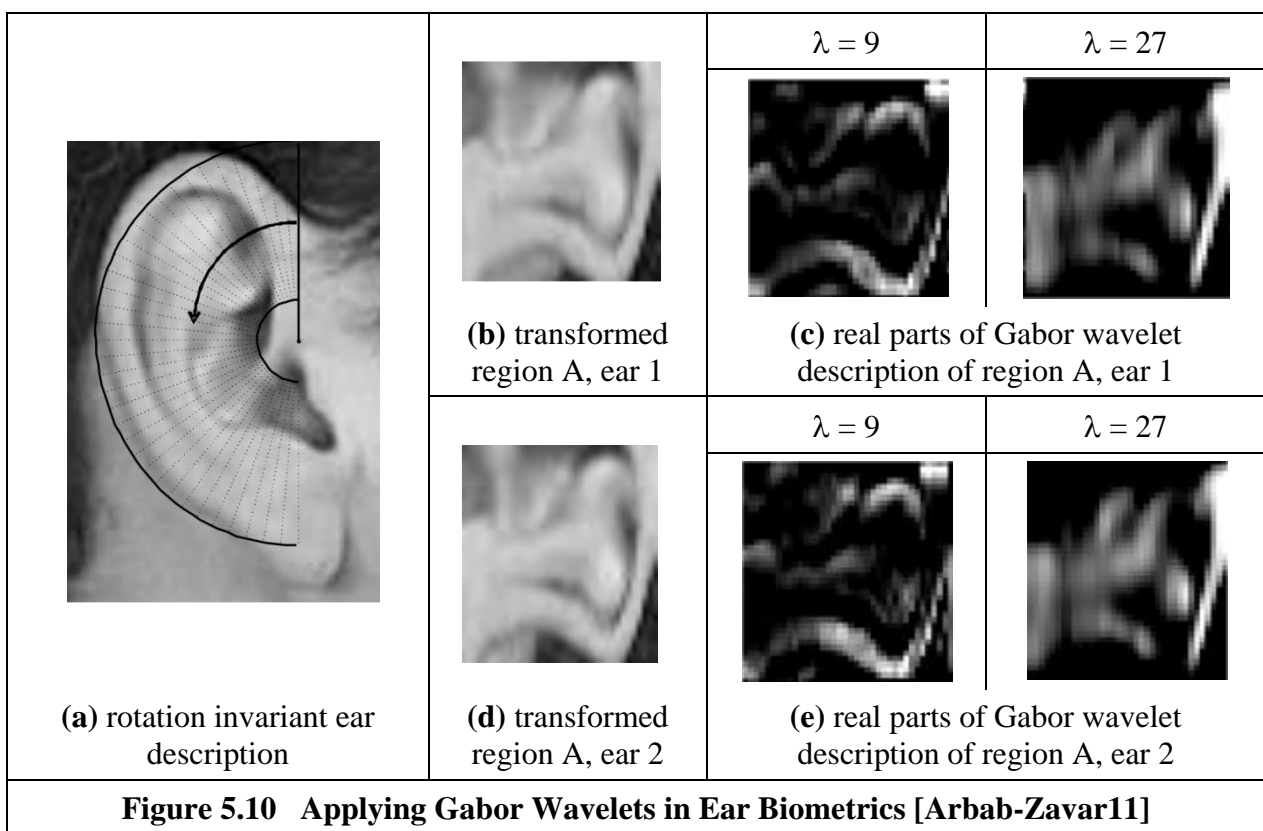
The *Viola Jones approach* essentially uses the form of Haar wavelets defined earlier in Section 2.7.3.2 as a basis for object detection [Viola01] which was later extended to be one of the most popular techniques for detecting human faces in images [Viola04]. Using rectangles to detect image features is an approximation, as there are features which can describe curved structure (derived using Gabor wavelets for example). It is however a fast approximation, since the features can be detected using the integral image approach. If we are to consider the face image in Figure 5.9(a) then the eyes are darker than the cheeks which are immediately below them, and the eyes are also darker than the bridge of the nose. As such, if we match the template in Figure 5.9(b) (this is the inverted form of the template in Figure 2.32(c)), then superimposing this template on the image at the position where it best matches the face leads to the image of Figure 5.9(c). The result is not too surprising, since it finds two dark parts between which there is a light part, and only the eyes and the bridge of the nose fit this description. (We could of course have a nostril template but a) you might be eating your dinner and b) when you look closely, quite a lot of the image fits the description “two small dark blobs with a light bit in the middle” – we can successfully find the eyes since they are a large structure fitting the template well.)



In this way we can define a series of templates (those in Figure 2.29) and match them to the image. In this way we can find the underlying shape. We need to sort the results to determine which are the most important and which collection best describes the face. That is where the approach advances to machine learning, which comes later in Chapter 12. For now, we rank the filters as to

their importance and then find shapes by using a collection of these low level features. The original technique was phrased around detecting objects [Viola01] and later phrased around finding human faces in particular [Viola04], and it has now become one of the stock approaches to detecting faces automatically within image data.

There are limitations to this approach, naturally. The use of rectangular features allows fast calculation, but does not match well structures that have a smoother contour. There are very many features possible in templates of any reasonable size, and so the set of features must be pruned so that the best are selected, and that is where the machine learning processes are necessary. In turn this implies that the feature extraction process needs training (in features and in data) – and that is similar indeed to human vision. There are demonstration versions of the technique and improvements include the use of rotated Haar features [Lienhart03] as well as inspiring many of the later approaches that collect parts for recognition. Should you be interested in more recent approaches to face detection, [Zafeiriou15] is well worth a look.



#### 5.4.1.2 Object Detection by Combinations of Parts

There have been many approaches which apply wavelets, and ones which are more complex than Haar wavelets, to detect objects by combinations of parts. These approaches allow for greater flexibility in the representation of the part since the wavelet can capture frequency, orientation and position (thus incurring the cost of computational complexity). A major advantage is that scale can be used, and objects can exist at, or persist over, a selection of scales. One such approach used wavelets as a basis for detecting people and cars [Schneiderman04] and even a door handle, thus emphasising generality of the approach. As with the Viola-Jones approach, this method requires deployment of machine learning techniques which then involves training. In this method the training occurs over different viewpoints to factor out the subject's – or object's – pose. The method groups input data into sets, and each set is a part. For a human face, the parts include the eyes, nose

and mouth and some un-named, but classified face regions, and these parts are (statistically) interdependent in most natural objects. Then machine learning techniques are used to maximise the likelihood of finding the parts correctly. Highly impressive results have been provided, though again the performance of the technique depends on training as well as on other factors. The main point of the technique here, is that wavelets can allow for greater freedom when representing an object as a collection of parts.

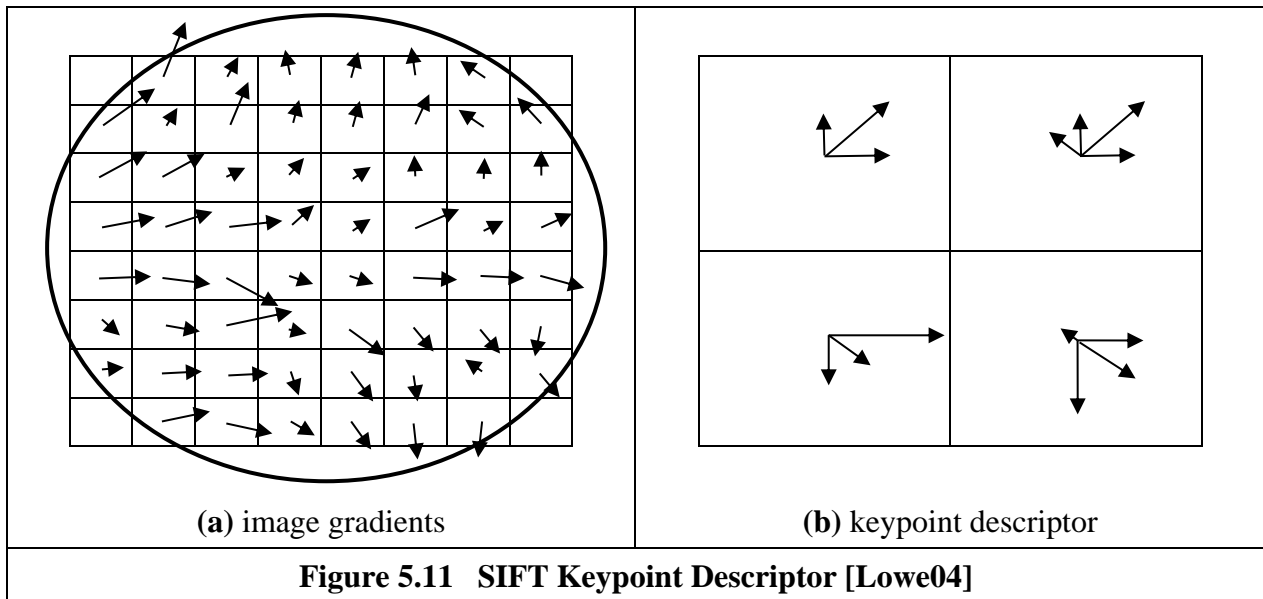
In our own research we have used Gabor wavelets in ear biometrics, where we can recognise a person's identity by analysis of the appearance of the ear [Hurley08]. It might be the ugliest biometric, but it also appears the most immune to effects of aging (except to enlarge): ears are fully formed at birth and change little throughout life, unlike the human face which changes rapidly as children grow teeth and then the general decline includes wrinkles and a few sags (unless a surgeon's expertise intervenes). In a way ears are like fingerprints, but the features are less clear. In our own research in biometrics, we have used Gabor wavelets to capture the ear's features [Arbab-Zavar11] in particular those relating to smooth curves. To achieve rotational invariance (in case a subject's head was tilted when the image was acquired), a radial scan was taken based on an ear's centre point, Figure 5.10(a), deriving the two transformed regions in Figure 5.10(b) and Figure 5.10(d) which are the same region for different images of the same ear. Then, these regions are transformed using a Gabor wavelet approach for which the real parts of the transform at two scales are shown in Figures 5.10(c) and Figure 5.10(e). Here, the detail is preserved at the short wavelength and the larger structures are detected at longer wavelengths. In both cases the prominent smooth structures are captured by the technique, leading to successful recognition of the subjects.

## 5.4.2 Distribution-Based Descriptors

### 5.4.2.1 Description by Interest Points (SIFT, SURF, BRIEF)

Lowe's *Scale Invariant Feature Transform* (SIFT) [Lowe04], Section 4.4.2.1, actually combines a scale invariant region detector with a descriptor which is based on the gradient distribution in the detected regions. The approach not only detects interest points but also provides a description for recognition purposes. The descriptor is represented by a 3D histogram of gradient locations and orientations and is created by first computing the gradient magnitude and orientation at each image point within the  $8 \times 8$  region around the keypoint location, as shown in Figure 5.11. These values are weighted by a Gaussian windowing function, indicated by the overlaid circle in Figure 5.11(a) wherein the standard deviation is chosen according the number of samples in the region (its width). This avoids fluctuation in the description with differing values of the keypoint's location and emphasises less the gradients that are far from the centre. These samples are then accumulated into orientation histograms summarizing the contents of the four  $4 \times 4$  sub regions, as shown in Figure 5.11(b), with the length of each arrow corresponding to the sum of the gradient magnitudes near that direction within the region. This involves a binning procedure as the histogram is quantised into a smaller number of levels (here eight compass directions are shown). The descriptor is then a vector of the magnitudes of the elements at each compass direction and in this case has  $4 \times 8 = 32$  elements. This figure shows a  $2 \times 2$  descriptor array derived from an  $8 \times 8$  set of samples and other arrangements are possible, such as  $4 \times 4$  descriptors derived from a  $16 \times 16$  sample array giving a 128 element descriptor. The final stage is to normalise the magnitudes so the description is illumination invariant. Given that SIFT has detected the set of keypoints and we have descriptions attached to each of those keypoints, we can then describe a shape by using the collection of parts detected by the SIFT technique. There is a variety of parameters that can be chosen within the approach, and the optimisation process is ably described [Lowe04] along with demonstration that the technique can be used to recognise objects, even in the presence of clutter and occlusion. An

improvement called *RootSIFT* [Arandjelović12] later changed the way distance/ similarity was measured to improve descriptonal capability and now appears to be in routine use.



The *SURF descriptor* [Bay08], Section 4.4.2.2, describes the distribution of the intensity content within the interest point neighbourhood, similar to SIFT (both approaches combine detection with description). In SURF, first a square region is constructed which is centred on an interest point and oriented along the detected orientation (detected via the Haar wavelets). Then, the description is derived from the Haar wavelet responses within the sub windows and the approach argues the approach “reduces the time for feature computation and matching, and has proven to simultaneously increase the robustness”.

The SIFT and SURF descriptors are large being vectors of 128 and 64 elements, respectively, and this can require much storage when millions of interest points are to be considered. The descriptors in *BRIEF* (Section 4.4.3) are longer but in number but comprise only of bits and the length is a natural compromise between speed and storage [Calonder10]. The bits are derived by pairwise intensity comparisons between smoothed patches thus predicating a need for sampling strategy and for smoothing (to reduce response to noise) where the comparison between intensities is

$$\tau(\mathbf{p1}, \mathbf{p2}) = \begin{cases} 1 & \mathbf{P}_{x1,y1} < \mathbf{P}_{x2,y2} \\ 0 & \text{otherwise} \end{cases} \quad (5.24)$$

The sequence of pairwise tests is stored as a bitstring which for  $N$  bits for comparison purposes

$$f_N(\mathbf{p}) = \sum_{i=1}^N 2^{i-1} \tau(\mathbf{p1}, \mathbf{p2}) \quad (5.25)$$

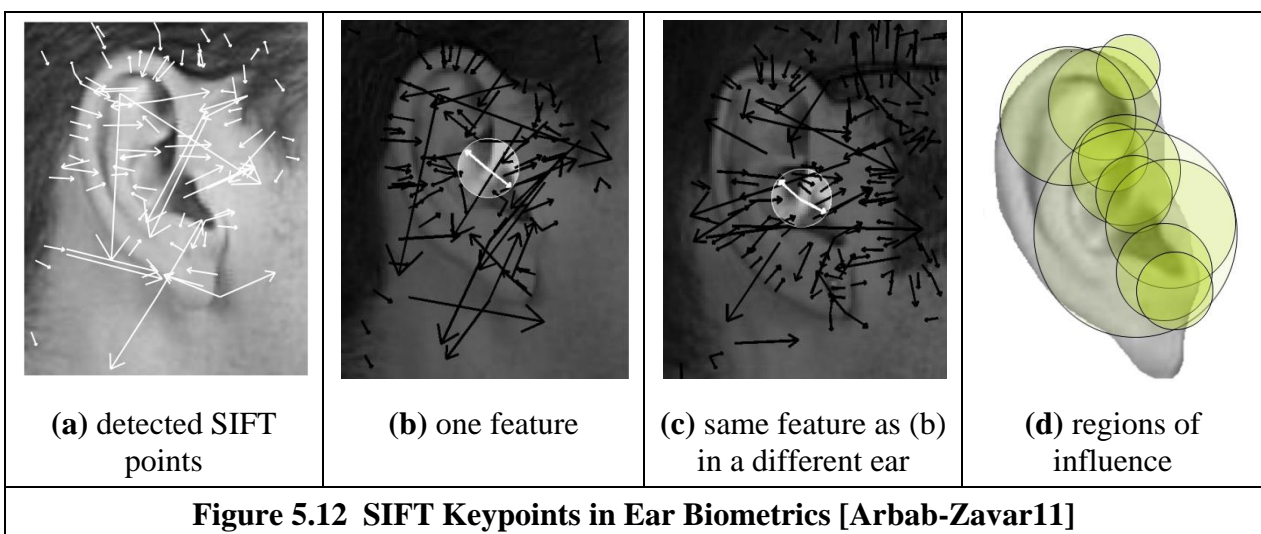
This was demonstrated to have good performance, especially in speed. Clearly it is very simple, and some of the learning power must be implicit in the learning techniques used within the pipeline. The *ORB* detector [Rublee11] is a very fast binary descriptor since it is based on BRIEF, and is rotation invariant and insensitive to noise. In ORB the direction of a patch depends on the direction from its centre to a corner and the rotation invariance is achieved by the corner description (which uses centralised moments (Section 7.3.2).

A major performance evaluation [Mikolajczyk05] compared the performance of descriptors computed for local interest regions and studied a number of operators, concerning in particular the

effects of geometric and affine transformations, for matching and recognition of the same object or scene.. The operators included a form of Gabor wavelets and SIFT and introduced the *Gradient Location and Orientation Histogram* (GLOH) which is an extension of the SIFT descriptor, and which appeared to offer better performance. The survey predated SURF and BRIEF and so they was not included. SIFT also performed well and there have been many applications of the SIFT approach for recognising objects in images, and the applications of SURF are burgeoning. One approach aimed to determine those key frames and shots of a video containing a particular object with ease and convenience of the Google search engine [Sivic03]. In this approach elliptical regions are represented by a 128-dimensional vector using the SIFT descriptor which was chosen by virtue of superior performance, especially when the object’s positions could vary by small amounts. From this, descriptions are constructed using machine learning techniques. A more recent comparison [Mukherjee15] highlights the all-round performance of ORB, SIFT and BRIEF, and is a good place to look.

In common with other object recognition approaches, we have deployed SIFT for ear biometrics [Arbab-Zavar11, Bustard10] to capture the description of an individual’s ear by a constellation of ear parts, again confirming that people appear unique by their ear. Here, the points detected are those that are significant across scales, and thus provide an alternative characterisation (to the earlier Gabor wavelet analysis in Section 5.4.1.2) of the ear’s appearance. Figure 5.12(a) shows the SIFT points detected within a human ear and Figures 5.12(b) and (c) show the same point (the crus of helix, no less) being detected in two different ears, and Figure 5.12(d) shows the domains of the SIFT points dominant in the ear biometrics procedure. Note that these points do not include the outer perimeter of the ear, which was described by Gabor wavelets. Recognition by the SIFT features was complemented by the Gabor features, as we derive descriptions of different regions, leading to the successful identification of the subjects by their ears. An extended discussion of how ears can be used as a biometric and the range of techniques that can be used for recognition is available [Hurley08].

As such we have concerned a topical area which continues to be of major current interest: CBIR is a common application of computer vision techniques. A recent study notes the prevalence now of deep learning “In recent years, the popularity of SIFT-based models seems to be overtaken by the convolutional neural network” citing performance issues [Zheng18]. However, there now many studies deploying interest point techniques for image matching, which show considerable performance capability. It is likely that the performance will be improved by technique refinement and analysis and therefore performance comparison and abilities will continue to develop



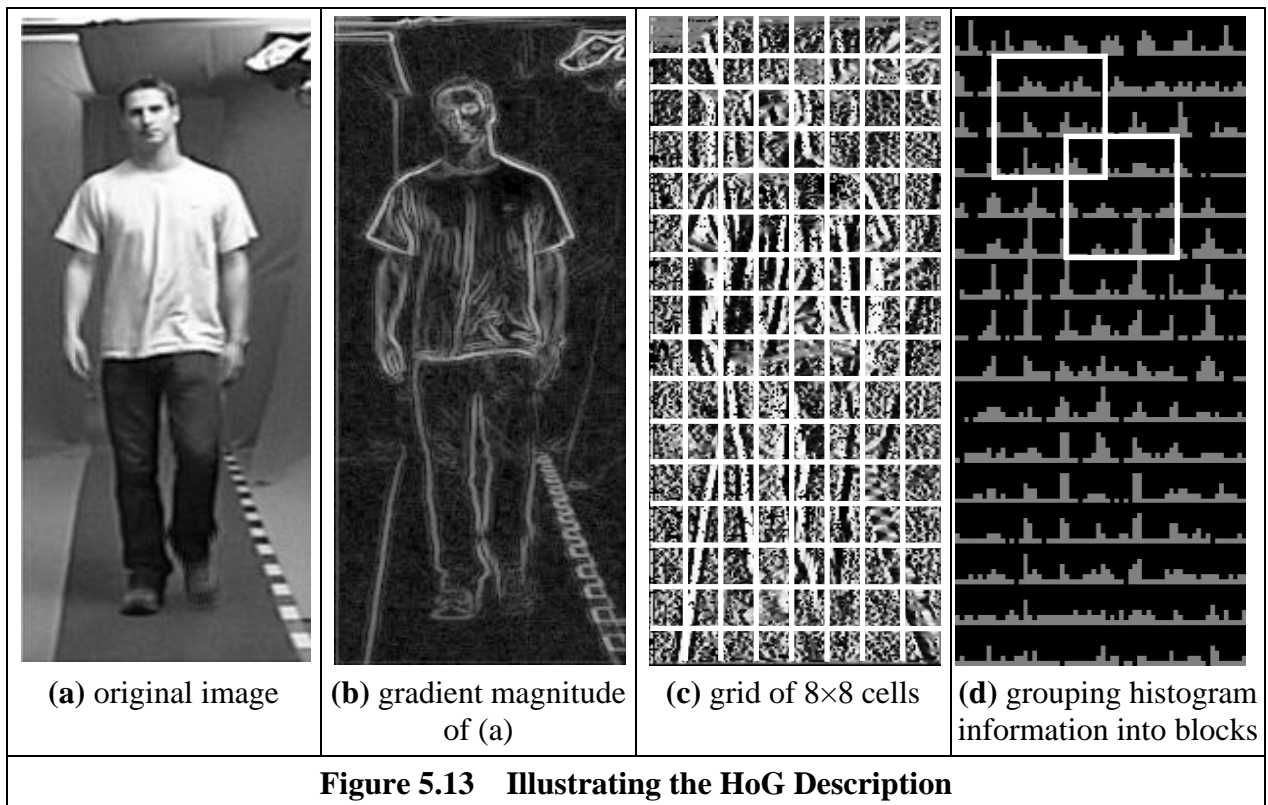
#### 5.4.2.2 Characterising Object Appearance and Shape

There has long been an interest in detecting pedestrians within scenes, more for automated surveillance analysis than for biometric purposes. The techniques have included use of Haar features and SIFT description. An approach called the *Histogram of Oriented Gradients (HoG)* [Dalal05] has received much interest. This captures edge or gradient structure that is very characteristic of local shape in a way which is relatively unaffected by appearance changes. Essentially, it forms a template and deploys machine learning approaches to expedite recognition, in an effective way. In this way it is an extension to describing objects by a histogram of the edge gradients.

First, edges are detected by the improved first order detector shown in Figure 4.4 and an edge image is created. Then, a vote is determined from a pixel's edge magnitude and direction and stored in a histogram. The direction is 'binned' in that votes are cast into roughly quantised histogram ranges and these votes are derived from cells, which group neighbourhoods of pixels. One implementation is to use  $8 \times 8$  image cells and to group these into  $20^\circ$  ranges (thus nine ranges within  $180^\circ$  of unsigned edge direction). Local contrast normalisation is used to handle variation in gradient magnitude due to change in illumination and contrast with the background, and this was determined to be an important stage. This normalisation is applied in blocks, eventually leading to the person's description which can then be learned by using machine learning approaches. Naturally there is a gamut of choices to be made, such as the choice of edge detection operator, inclusion of operator, cell size, the number of bins in the histogram and use of full  $360^\circ$  edge direction. Robustness is achieved in that noise or other effects should not change the histograms much: the filtering is done at the description stage rather than at the image stage (as with wavelet-based approaches).

The process of building the HoG description is illustrated in Figure 5.13 where (a) is the original image; (b) is the gradient magnitude constructed from the absolute values of the improved first order difference operator; (c) is the grid of  $8 \times 8$ , superimposed on the edge direction image; and (d) illustrates the  $3 \times 3$  (rectangular) grouping of the cells, superimposed on the histograms of gradient data. There is a rather natural balance between the grid size and the size of the grouping arrangements, though these can be investigated in application. Components of the walking person can be seen especially in the preponderance of vertical edge components in the legs and thorax. The grouping and normalisation of these data lead to the descriptor which can be deployed so as to detect humans/ pedestrians in static images.

The approach is not restricted to detecting pedestrians since it can be trained to detect different shapes and it has been applied elsewhere. Given there is much interest in speed of computation, rather unexpectedly a Fast HoG was to appear soon after the original HoG [Zhu06] and which claims 30 fps capability. An alternative approach, and one which confers greater generality – especially with humans – is to include the possibility of deformation, as will be covered in Section 6.2.



Essentially, these approaches can achieve fast extraction by decomposing a shape into its constituent parts. Clearly one detraction of the techniques is that if you are to change implementation - or to detect other objects - then this requires construction of the necessary models and parts, and that can be quite demanding. In fact, it can be less demanding to include shape, and as template matching can give a guaranteed result, another class of approaches is to reformulate template matching so as to improve speed. That is the Hough Transform, coming next.

## 5.5 Hough Transform (HT)

### 5.5.1 Overview

The *Hough Transform* (HT) [Hough62] is a technique that locates shapes in images. In particular, it has been used to extract lines, circles and ellipses (or conic sections). In the case of lines, its mathematical definition is equivalent to the Radon transform [Deans81]. The HT was introduced by Hough [Hough62] and then used to find bubble tracks rather than shapes in images. However, Rosenfeld noted its potential advantages as an image processing algorithm [Rosenfeld69]. The HT was thus implemented to find lines in images [Duda72] and it has been extended greatly, since it has many advantages and many potential routes for improvement. Its prime advantage is that it can deliver the same result as that for template matching, but faster [Princen92] [Sklansky78][Stockman77]. This is achieved by a reformulation of the template matching process, based on an *evidence gathering* approach where the evidence is the votes cast in an accumulator array. The HT implementation defines a mapping from the image points into an accumulator space (Hough space). The mapping is achieved in a computationally efficient manner, based on the function that describes the target shape. This mapping requires much less computational resources than template matching. However, it still requires significant storage and high computational requirements. These problems are addressed later, since they give focus for the continuing



development of the HT. However, the fact that the HT is equivalent to template matching has given sufficient impetus for the technique to be amongst the most popular of all existing shape extraction techniques.

### 5.5.2 Lines

We will first consider finding lines in an image. In a Cartesian parameterisation, collinear points in an image with co-ordinates  $(x, y)$  are related by their slope  $m$  and an intercept  $c$  according to:

$$y = mx + c \quad (5.26)$$

This equation can be written in homogeneous form as

$$Ay + Bx + 1 = 0 \quad (5.27)$$

where  $A = -1/c$  and  $B = m/c$ . Thus, a line is defined by giving a pair of values  $(A, B)$ . However, we can observe a symmetry in the definition in Equation (5.27). This equation is symmetric since a pair of co-ordinates  $(x, y)$  also defines a line in the space with parameters  $(A, B)$ . That is, Equation (5.27) can be seen as the equation of a line for fixed co-ordinates  $(x, y)$  or as the equation of a line for fixed parameters  $(A, B)$ . Thus, pairs can be used to define points and lines simultaneously [Aguado00a]. The HT gathers evidence of the point  $(A, B)$  by considering that all the points  $(x, y)$  define the same line in the space  $(A, B)$ . That is, if the set of collinear points  $\{(x_i, y_i)\}$  defines the line  $(A, B)$ , then

$$Ay_i + Bx_i + 1 = 0 \quad (5.28)$$

This equation can be seen as a system of equations and it can simply be rewritten in terms of the Cartesian parameterisation as

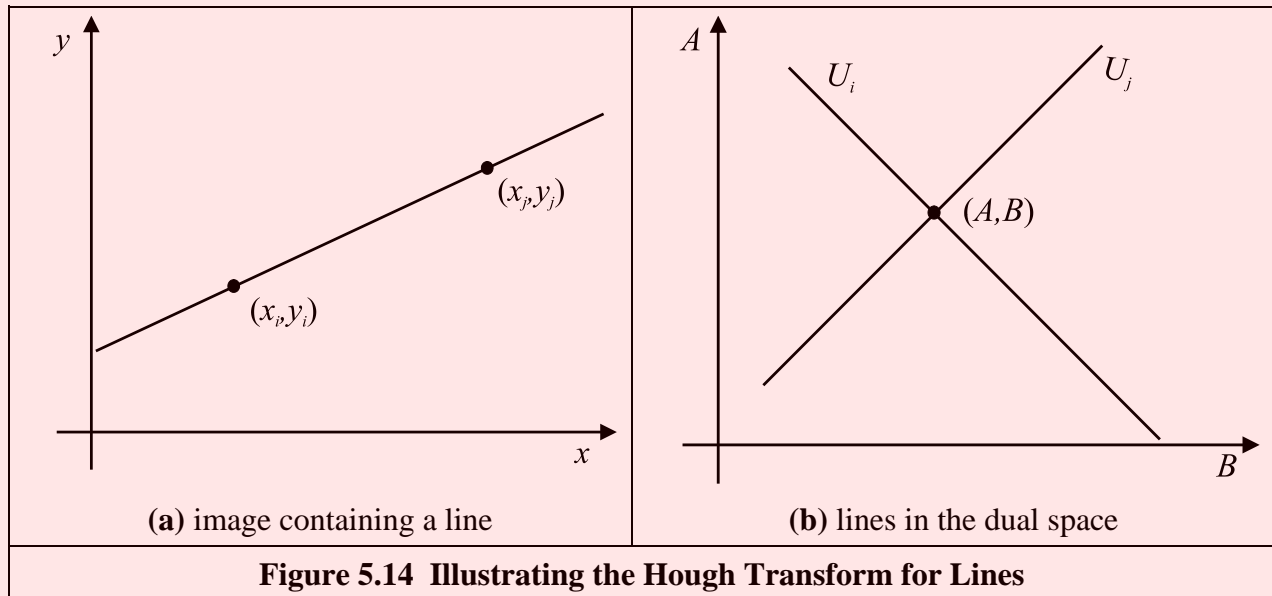
$$c = -x_i m + y_i \quad (5.29)$$

Thus, to determine the line we must find the values of the parameters  $(m, c)$  (or  $(A, B)$  in homogeneous form) that satisfy Equation (5.29) (or 5.28, respectively). However, we must notice that the system is generally over-determined. That is, we have more equations than unknowns. Thus, we must find the solution that comes close to satisfying all the equations simultaneously. This kind of problem can be solved, for example, using linear least-squares techniques. The HT uses an evidence gathering approach to provide the solution. Notice that if  $x$  is zero in Equation (5.29), then  $y_i = c$ . That is,  $c$  is the intersection of the line with the  $y$  axis.

The relationship between a point  $(x_i, y_i)$  in an image and the line given in Equation (5.29) is illustrated in Figure 5.14. The points  $(x_i, y_i)$  and  $(x_j, y_j)$  in Figure 5.14(a) define the lines  $U_i$  and  $U_j$  in Figure 5.14(b), respectively. All the collinear elements in an image will define dual lines with the same concurrent point  $(A, B)$ . This is independent of the line parameterisation used. The HT solves it in an efficient way by simply counting the potential solutions in an accumulator array that stores the evidence, or votes. The count is made by tracing all the dual lines for each point  $(x_i, y_i)$ . Each point in the trace increments an element in the array, thus the problem of line extraction is transformed to the problem of locating a maximum in the accumulator space. This strategy is robust and has demonstrated to be able to handle noise and occlusion.

The axes in the dual space represent the parameters of the line. In the case of the Cartesian parameterisation  $m$  can actually take an infinite range of values, since lines can vary from horizontal to vertical. Since votes are gathered in a discrete array, then this will produce bias errors. It is possible to consider a range of votes in the accumulator space that cover all possible values.

This corresponds to techniques of antialiasing and can improve the gathering strategy [Brown83][Kiryati91].



The implementation of the HT for lines is given in Code 5.4. The implementation gathers evidence for each point  $(x, y)$  by drawing a line obtained by iterating over the parameter  $m$ . Thus, the iteration generates points  $(m, c)$  in the accumulator. It is important to observe that Equation (5.29) is not suitable for implementation since the parameter  $c$  can take an infinite range of values. In order to avoid this, the implementation uses two accumulator arrays. One accumulator is used to gather lines with slopes between  $-45^\circ$  and  $45^\circ$  and another accumulator for lines with slopes between  $45^\circ$  and  $135^\circ$ . The main loop in the implementation iterates over a range from  $0^\circ$  to  $90^\circ$ . Thus, each accumulator has  $90^\circ$  range and together both accumulators consider the full  $180^\circ$ . For the first accumulator lines are horizontal, so we can compute the intercept  $c$  according to Equation (5.29). In this case, the value  $c$  is the intersection of the line with the  $y$  axis. Notice that the size of the intercept is defined as twice the image size since the intercept of lines in images is not limited by the height of the image. In the second case, the lines are vertical, so the code computes the intersection with they  $x$  axis. As such, the line is then defined as  $x = y/m + c$ . Notice that when the tangent of  $m$  is computed in the code,  $m$  is limited to values between  $-45^\circ$  and  $45^\circ$ .

The implementation in Code 5.4 introduces weighting in the accumulator process. This weight is computed as the difference between the position of the accumulator and the position of the point in the voting line. This is introduced to reduce discretization errors caused by the fact that we are drawing a continuous curve into a discrete array. Instead of increasing a single entry for each point in the line, the code increments two positions by an amount proportional to the distance to the continuous line. The use of weights produces accumulators with less noise.

```

# Gather evidence
for x,y in itertools.product(range(0, width), range(0, height)):
    if magnitude[y,x] != 0:
        for m in range(0,90):

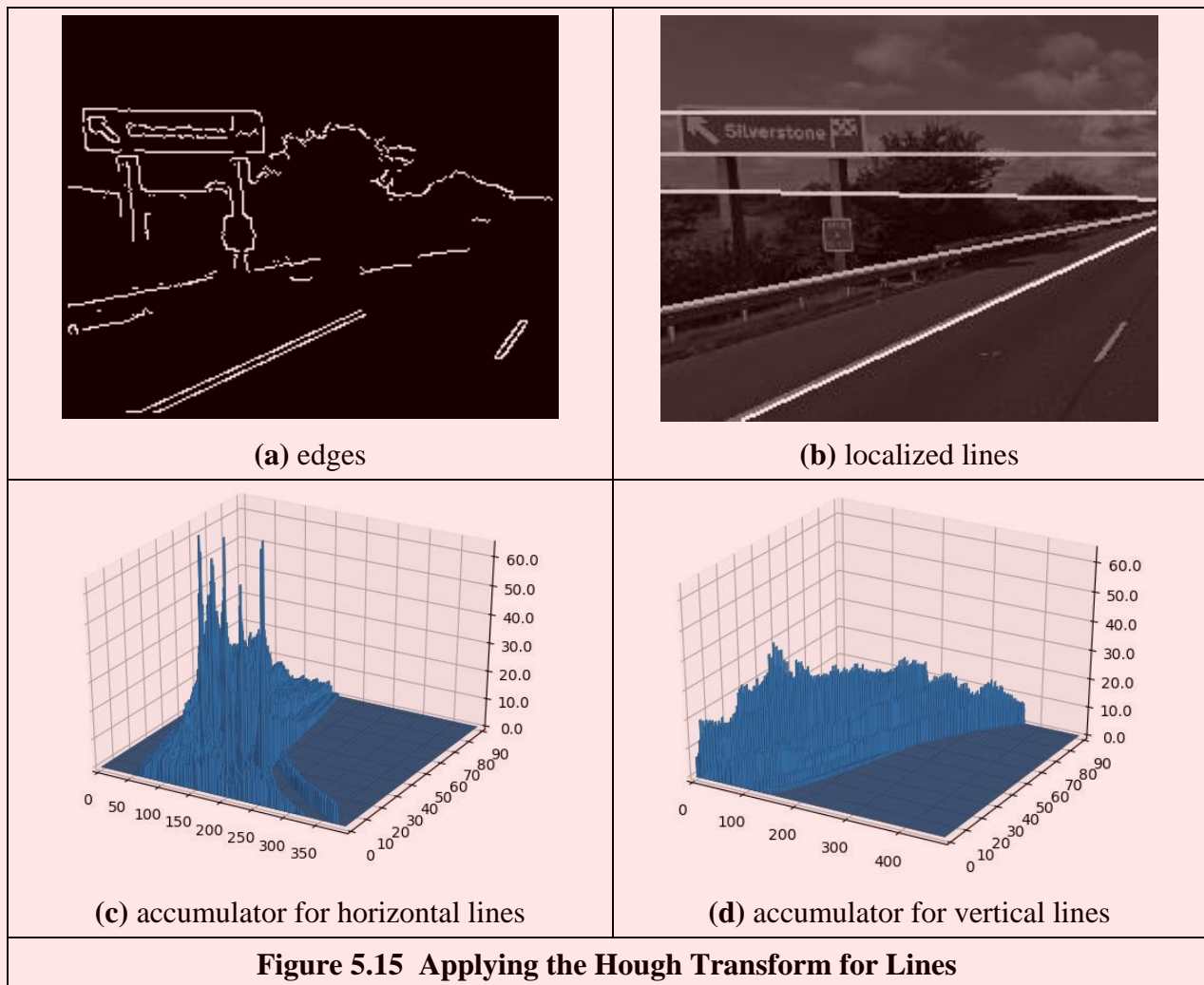
            # Lines between -45 and 45 degrees
            angle = ((-45 + m) * math.pi) / 180.0
            c = y - math.tan(angle) * x
            bucket = int(c)
            if bucket > 0 and bucket < 2*height - 1:
                weight = c - int(c)
                accHorizontal[m, bucket] += (1.0 - weight)
                accHorizontal[m, bucket+1] += weight

            # Lines between 45 and 135 degrees
            angle = ((45.0 + m) * math.pi) / 180.0
            c = x - y / math.tan(angle)
            bucket = int(c)
            if bucket > 0 and bucket < 2*width - 1:
                weight = c - int(c)
                accVertical[m, bucket] += (1.0 - weight)
                accVertical[m, bucket+1] += weight

```

#### Code 5.4 Implementing the Hough transform for lines

Figure 5.15 shows an example of the location results obtained by using the HT implemented in Code 5.4. Figure 5.15(a) shows the edges used in the accumulator process. Figures 5.15(c) and (d) show the vertical and horizontal accumulators. The position of each peak defines the parameters  $(m, c)$  of a line. The magnitude of the peaks is proportional to the number of pixels in the line from which it was generated. So small peaks represent small lines in the image. Figure 5.15(b) shows the lines represented by the maxima in the accumulators. These lines were drawn by iterating the values  $(x, y)$  for a peak  $(m, c)$  in the accumulator. Here, we can see that the parameters describing the lines have been extracted well. Note that the end points of the lines are not delivered by the HT, only the parameters that describe them. You have to go back to the image to obtain line length.



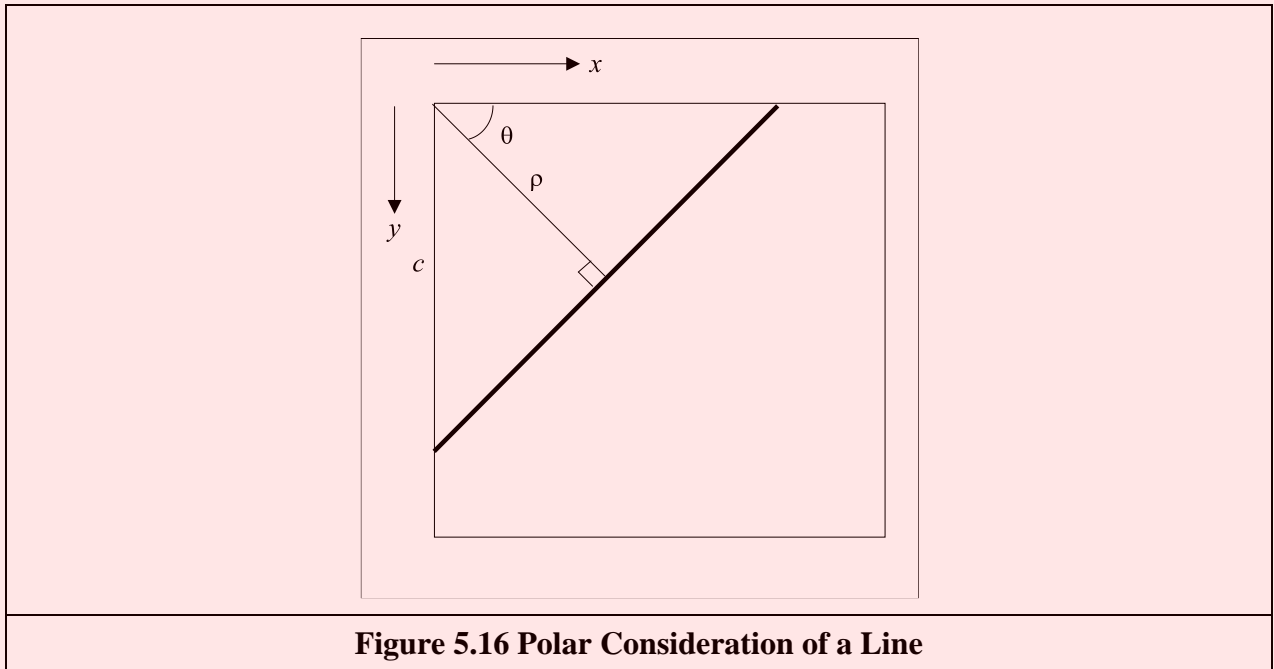
We can see that the HT delivers a correct response, correct estimates of the parameters used to specify the line, so long as the number of collinear points along that line exceeds the number of collinear points on any other line in the image. As such, the Hough transform has the same properties in respect of noise and occlusion, as with template matching. However, the non-linearity of the parameters and the discretisation produce noisy accumulators. A major problem in implementing the basic HT for lines is the definition of an appropriate accumulator space. In application, *Bresenham's* line drawing algorithm [Bresenham65] can be used to draw the lines of votes in the accumulator space. This ensures that lines of connected votes are drawn as opposed to use of Equation (5.29) that can lead to gaps in the drawn line. Also, *backmapping* [Gerig86] can be used to determine exactly which edge points contributed to a particular peak. Backmapping is an inverse mapping from the accumulator space to the edge data and can allow for shape analysis of the image by removal of the edge points which contributed to particular peaks, and then by re-accumulation using the HT. Note that the computational cost of the HT depends on the number of edge points ( $n_e$ ) and the length of the lines formed in the parameter space ( $l$ ), giving a computational cost of  $O(n_e l)$ . This is considerably less than that for template matching, given earlier as  $O(N^2 m^2)$ .

One way to avoid the problems of the Cartesian parameterisation in the HT is to base the mapping function on an alternative parameterisation. One of the most proven techniques is called the *foot-of-normal* parameterisation. This parameterises a line by considering a point  $(x, y)$  as a function of an angle normal to the line, passing through the origin of the image. This gives a form

of the HT for lines known as the *polar HT for lines* [Duda72]. The point where this line intersects the line in the image is given by

$$\rho = x \cos(\theta) + y \sin(\theta) \tag{5.30}$$

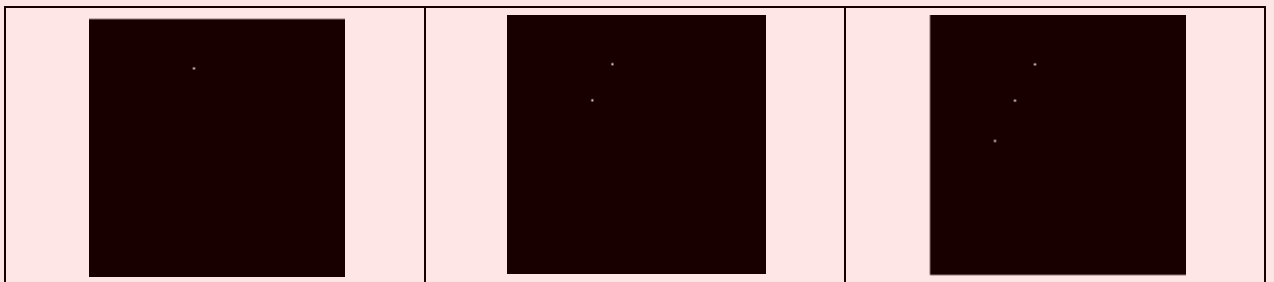
where  $\theta$  is the angle of the line normal to the line in an image and  $\rho$  is the length between the origin and the point where the lines intersect, as illustrated in Figure 5.16.

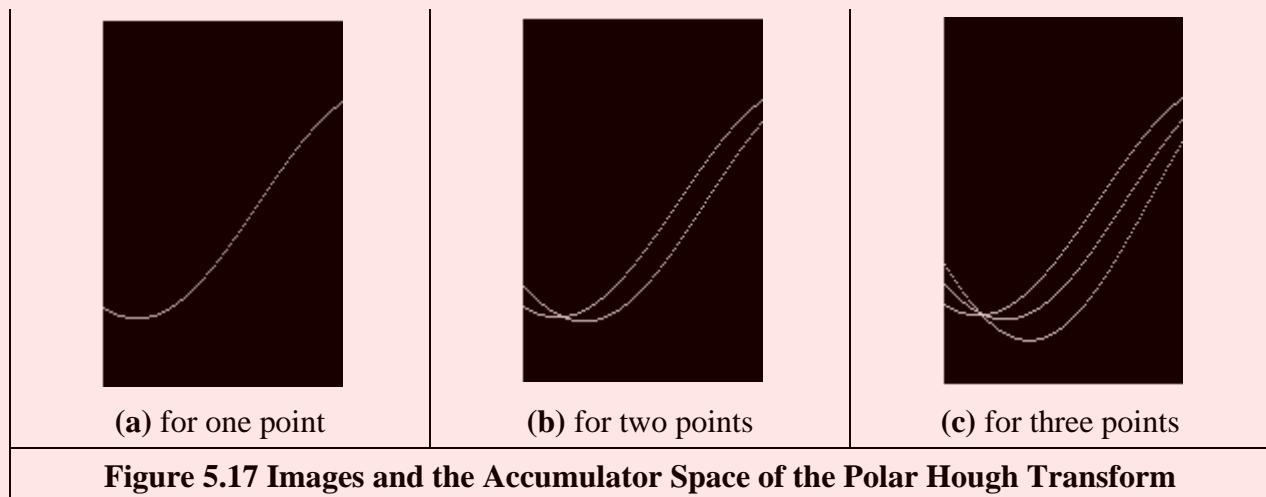


By recalling that two lines are perpendicular if the product of their slopes is -1, and by considering the geometry of the arrangement in Figure 5.16, we obtain

$$c = \frac{\rho}{\sin(\theta)} \quad m = -\frac{1}{\tan(\theta)} \tag{5.31}$$

By substitution in Equation (5.26) we obtain the polar form, Equation (5.30). This provides a different mapping function: votes are now cast in a sinusoidal manner, in a 2D accumulator array in terms of  $\theta$  and  $\rho$ , the parameters of interest. The advantage of this alternative mapping is that the values of the parameters  $\theta$  and  $\rho$  are now bounded to lie within a specific range. The range for  $\theta$  is within  $180^\circ$ ; the possible values of  $\rho$  are given by the image size, since the maximum length of the line is  $\sqrt{2} \times N$ , where  $N$  is the (square) image size. The range of possible values is now fixed, so the technique is practicable.



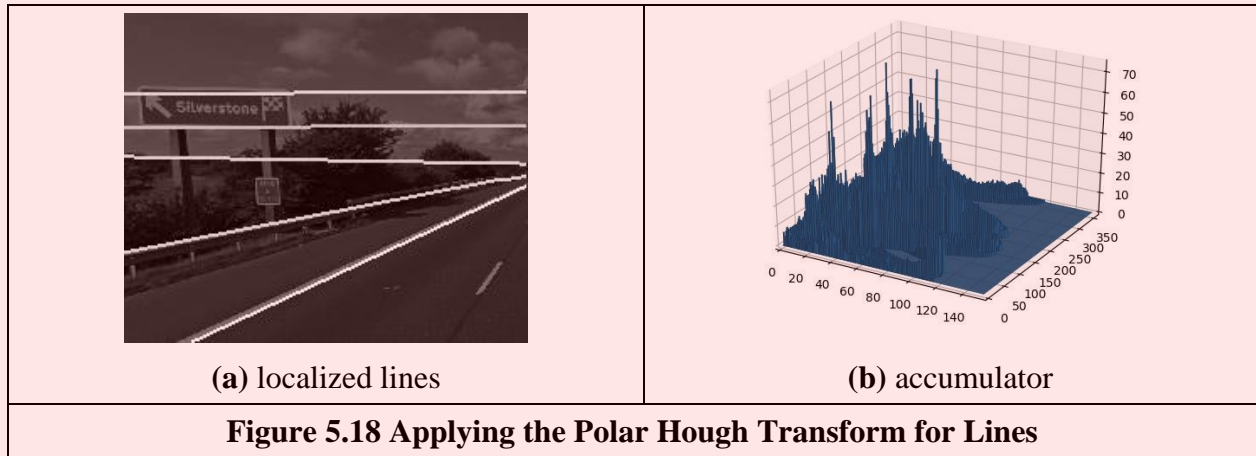


As the voting function has now changed, we shall draw different loci in the accumulator space. In the conventional HT for lines, a straight line mapped to a straight line as in Figure 5.17. In the polar HT for lines, points map to curves in the accumulator space. This is illustrated in Figure 5.17 which shows the polar HT accumulator spaces for (a) one, (b) two and (c) three points, respectively. For a single point in the upper row of Figure 5.17(a) we obtain a single curve shown in the lower row of Figure 5.17(a). For two points we obtain two curves, which intersect at a position which describes the parameters of the line joining them, Figure 5.17(b). An additional curve obtains for the third point and there is now a peak in the accumulator array containing three votes, Figure 5.17(c).

```
# Gather evidence
for x,y in itertools.product(range(0, width), range(0, height)):
    if magnitude[y,x] != 0:
        for m in range(0,360):
            angle = (m * math.pi) / 180.0
            r = (x-cx) * math.cos(angle) + (y-cy) * math.sin(angle)
            bucket = int(r)
            if bucket > 0 and bucket < maxLenght - 1:
                weight = r - int(r)
                accumulator[m, bucket] += (1.0 - weight)
                accumulator[m, bucket+1] += weight
```

**Code 5.5 Implementation of the Polar Hough Transform for Lines**

The implementation of the polar HT for lines is presented in Code 5.5. The accumulator array is a set of 360 bins for value of  $\theta$  in the range  $0^\circ$  to  $180^\circ$ , and for values of  $\rho$  in the range 0 to  $\sqrt{N^2 + M^2}$ , where  $N \times M$  is the picture size. Then, for image (edge) points greater than a chosen threshold, the angle relating to the bin size is evaluated (as radians in the range 0 to  $\pi$ ) and then the value of  $\rho$  is evaluated using Equation (5.30) and the appropriate accumulator cell is incremented so long as the parameters are within the range. Similar to the HT for lines, a weight is used to decrease the noise caused by the discretization of the accumulator. The accumulator array obtained by applying this implementation to the image in Figure 5.4 is shown in Figure 5.18 and shows a single accumulator that is used to gather data about the lines in the image. Figure 5.18(b) shows peaks that define the lines in Figure 5.18(a). Using a single accumulator makes the polar implementation far more practicable than the earlier Cartesian version.

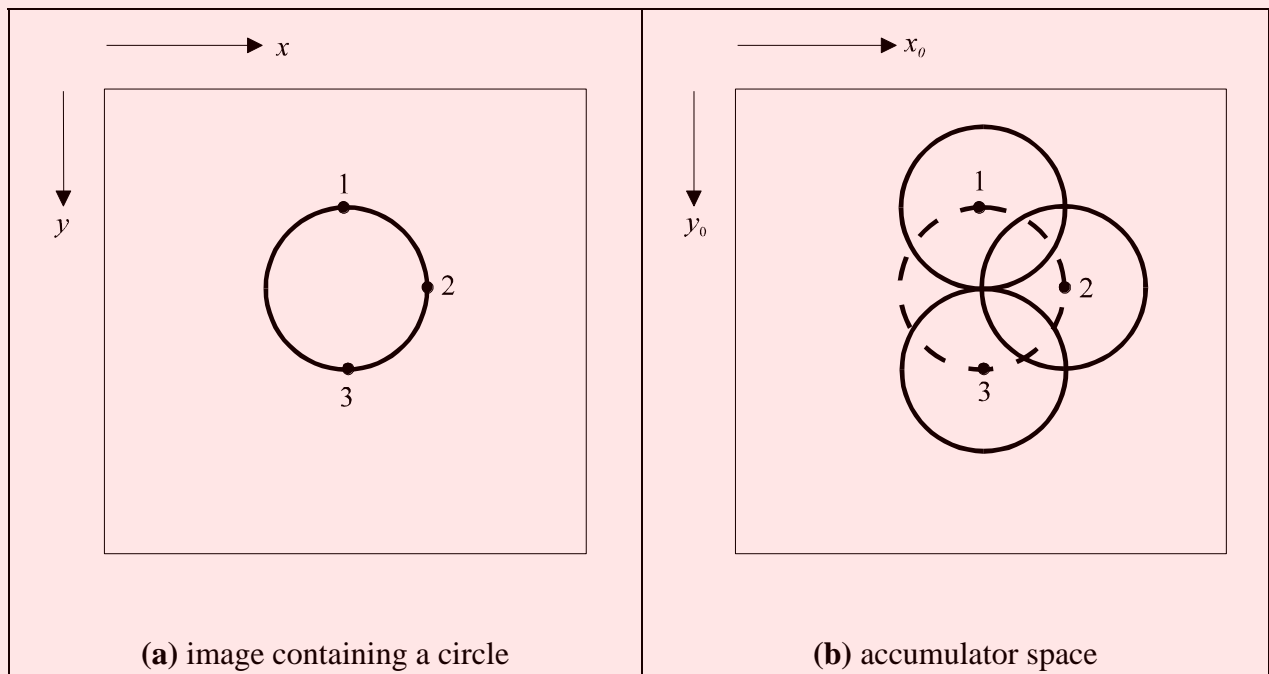


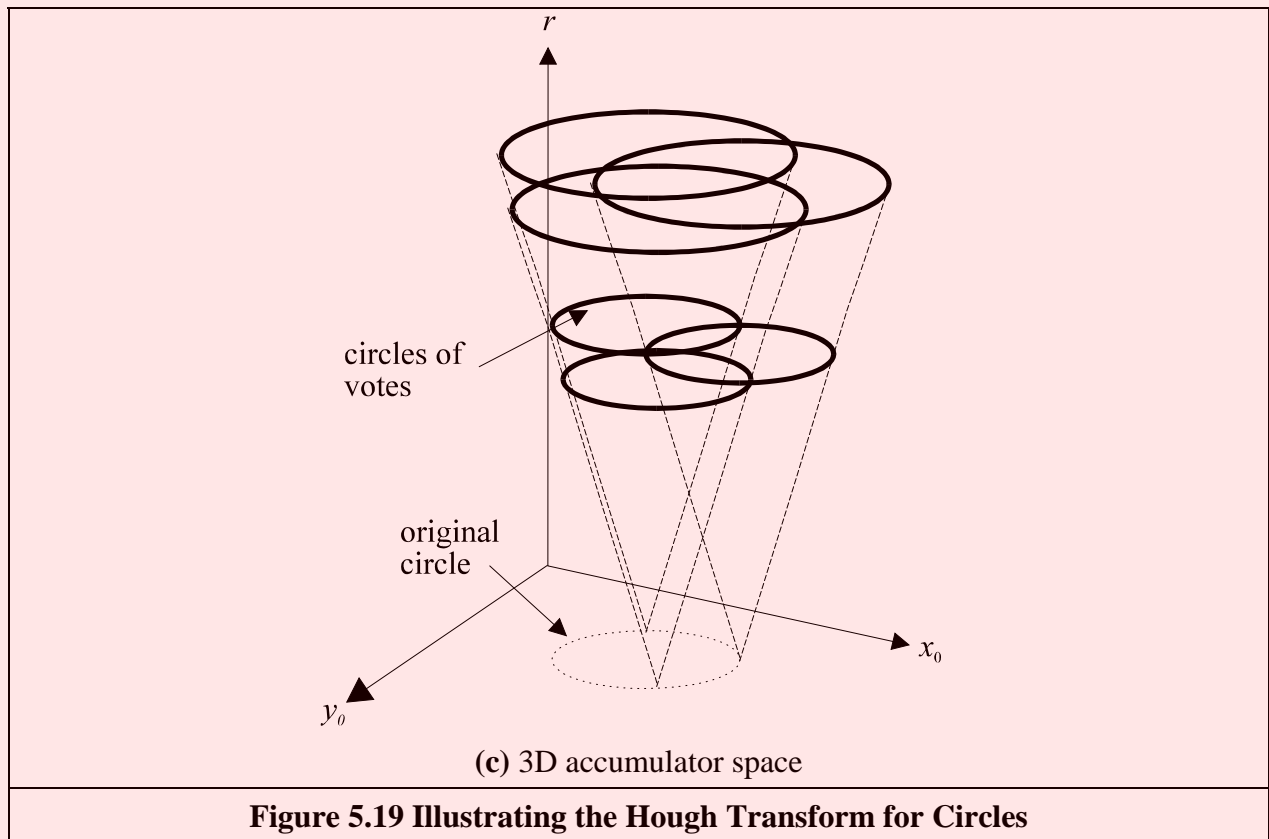
### 5.5.3 HT for Circles

The HT can be extended by replacing the equation of the curve in the detection process. The equation of the curve can be given in explicit or parametric form. In explicit form, the HT can be defined by considering the equation for a circle given by

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{5.32}$$

This equation defines a locus of points  $(x, y)$  centred on an origin  $(x_0, y_0)$  and with radius  $r$ . This equation can again be visualised in two dual ways: as a locus of points  $(x, y)$  in an image, or as a locus of points  $(x_0, y_0)$  centred on  $(x, y)$  with radius  $r$ .





**Figure 5.19 Illustrating the Hough Transform for Circles**

Figure 5.19 illustrates this dual definition. Each edge point in Figure 5.19(a) defines a set of circles in the accumulator space. These circles are defined by all possible values of the radius and they are centred on the co-ordinates of the edge point. Figure 5.19(b) shows three circles defined by three edge points. These circles are defined for a given radius value. Actually, each edge point defines circles for the other values of the radius. This implies that the accumulator space is three dimensional (for the three parameters of interest) and that edge points map to a cone of votes in the accumulator space. Figure 5.19(c) illustrates this accumulator. After gathering evidence of all the edge points, the maximum in the accumulator space again corresponds to the parameters of the circle in the original image. The procedure of evidence gathering is the same as that for the HT for lines, but votes are generated in cones, according to Equation (5.32).

Equation (5.32) can be defined in parametric form as

$$x = x_0 + r \cos(\theta) \quad y = y_0 + r \sin(\theta) \quad (5.33)$$

The advantage of this representation is that it allows us to solve for the parameters. Thus, the HT mapping is defined by

$$x_0 = x - r \cos(\theta) \quad y_0 = y - r \sin(\theta) \quad (5.34)$$

These equations define the points in the accumulator space (Figure 5.19(b)) dependent on the radius  $r$ . Note that  $\theta$  is not a free parameter, but defines the trace of the curve. The trace of the curve (or surface) is commonly referred to as the *point spread function*.

The implementation of the HT for circles is shown in Code 5.6. This is similar to the HT for lines, except that the voting function corresponds to that in Equation (5.34) and the accumulator space is actually 3D, in terms of the centre parameters and the radius. In the implementation the parameter `maxr` defines the maximum radius size. A circle of votes is generated by varying `theta` (Python and Matlab do not allow Greek symbols!) from  $0^\circ$  to  $360^\circ$ . The discretisation of `theta` controls the granularity of voting, too small increments give very fine coverage of the parameter



space, if it is too large then it produces a sparse coverage. The accumulator space, `accumulator` (initially zero), is incremented only for points whose co-ordinates lie within the specified range (in this case the centre cannot lie outside the original image).

```
function accum = circle_hough(image, maxr)

%get dimensions
[rows,cols]=size(image);

%set a start radius
startr=10;

%set the output image to black (0)
accum(1:rows,1:cols,1:(maxr-startr+1))=0;

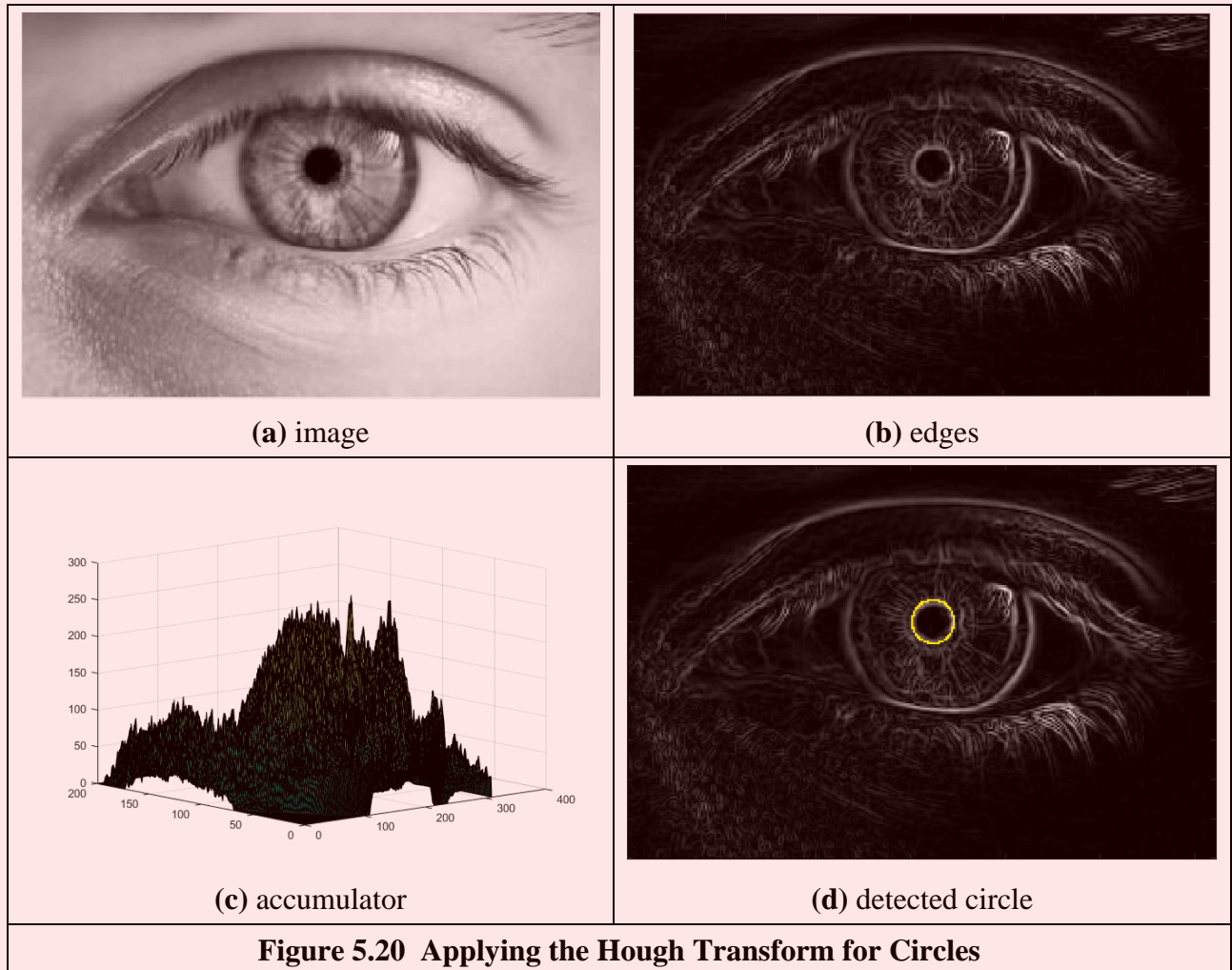
%and an output image
accum(1:rows,1:cols)=0;

for x = 1:cols %address all columns
    for y = 1:rows %address all rows
        for rad = startr:maxr
            if image(y,x)>80
                for theta=1:360
                    xdash=round(rad*cos(theta*pi/180));
                    ydash=round(rad*sin(theta*pi/180));
                    if ((x+xdash)<cols) && ((x+xdash)>0) &&...
                        ((y+ydash)<rows) && ((y+ydash)>0))
                        accum(y+ydash,x+xdash,rad-startr+1)=...
                            accum(y+ydash,x+xdash,rad-startr+1)+1;
                    end
                end
            end
        end
    end
end

parameters=get_max(accum);
```

### Code 5.6 Implementating the Hough Transform for Circles

The application of the HT for circles is illustrated in Figure 5.20. Figure 5.20(a) shows an image of an eye. Figure 5.20(b) shows the edges computed from the image. The edges are well defined and there are many edges produced by noise (as ever). The result of the HT process is shown in Figure 5.20(c) aimed to, and succeeding to, detect the border of the pupil. The peak of the accumulator space represents the centre of the circle, though it only just exists as a peak in this noisy image. Many votes exist away from the circle's centre, though these background votes are less than the actual peak. Figure 5.20(d) shows the detected circle defined by the parameters with the maximum in the accumulator. The HT will detect the circle (provide the right result) as long as more points are in a circular locus described by the parameters of the target circle than there are on any other circle. This is exactly the same performance as for the HT for lines, as expected, and is consistent with the result of template matching. Note that the HT merely finds the circle with the maximum number of points (in Figure 5.20(d) a small range of radii was given to detect the perimeter of the pupil); it is possible to include other constraints to control the circle selection process, such as gradient direction for objects with known illumination profile. In the case of the human eye, the (circular) iris is usually darker than its white surroundings.



**Figure 5.20** Applying the Hough Transform for Circles

In application code, *Bresenham's algorithm* for discrete circles [Bresenham77] can be used to draw the circle of votes, rather than use the polar implementation of Equation (5.34). This ensures that the complete locus of points is drawn and avoids the need to choose a value for increase in the angle used to trace the circle. Bresenham's algorithm can be used to generate the points in one octant, since the remaining points can be obtained by reflection. Again, backmapping can be used to determine which points contributed to the extracted circle.

Figure 5.20 also shows some of the difficulties with the HT, namely that it is essentially an implementation of template matching, and does not use some of the richer stock of information available in an image. For example, we might know constraints on size: the largest size an iris or pupil would be in the image. Also, we know some of the topology: the eye region contains two ellipsoidal structures with a circle in the middle. We might also know brightness information: the pupil is darker than the surrounding iris. These factors can be formulated as constraints on whether edge points can vote within the accumulator array. A simple modification is to make the votes proportional to edge magnitude, in this manner, points with high contrast will generate more votes and hence have more significance in the voting process. In this way, the feature extracted by the HT can be arranged to suit a particular application.

#### 5.5.4 HT for Ellipses

Circles are very important in shape detection since many objects have a circular shape. However, because of the camera's viewpoint, circles do not always look like circles in images. Images are formed by mapping a shape in 3D space into a plane (the image plane). This mapping performs a

perspective transformation. In this process, a circle is deformed to look like an ellipse. We can define the mapping between the circle and an ellipse by a similarity transformation. That is,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (5.35)$$

where  $(x', y')$  define the co-ordinates of the circle in Equation (5.33),  $\rho$  represents the orientation,  $(S_x, S_y)$  a scale factor and  $(t_x, t_y)$  a translation. If we define

$$\begin{aligned} a_0 = t_x \quad a_x = S_x \cos(\rho) \quad b_x = S_y \sin(\rho) \\ b_0 = t_y \quad a_y = -S_x \sin(\rho) \quad b_y = S_y \cos(\rho) \end{aligned} \quad (5.36)$$

then the circle is deformed into

$$\begin{aligned} x &= a_0 + a_x \cos(\theta) + b_x \sin(\theta) \\ y &= b_0 + a_y \cos(\theta) + b_y \sin(\theta) \end{aligned} \quad (5.37)$$

This equation corresponds to the polar representation of an ellipse. This polar form contains six parameters  $(a_0, b_0, a_x, b_x, a_y, b_y)$  that characterise the shape of the ellipse.  $\theta$  is not a free parameter and it only addresses a particular point in the locus of the ellipse (just as it was used to trace the circle in Equation 5.34). However, one parameter is redundant since it can be computed by considering the orthogonality (independence) of the axes of the ellipse (the product  $a_x b_x + a_y b_y = 0$  which is one of the known properties of an ellipse). Thus, an ellipse is defined by its centre  $(a_0, b_0)$  and three of the axis parameters  $(a_x, b_x, a_y, b_y)$ . This gives five parameters which is intuitively correct since an ellipse is defined by its centre (2 parameters), its size along both axes (2 more parameters) and its rotation (1 parameter). In total this states that 5 parameters describe an ellipse, so our three axis parameters must jointly describe size and rotation. In fact, the axis parameters can be related to the orientation and the length along the axes by

$$\tan(\rho) = \frac{a_y}{a_x} \quad a = \sqrt{a_x^2 + a_y^2} \quad b = \sqrt{b_x^2 + b_y^2} \quad (5.38)$$

where  $(a, b)$  are the axes of the ellipse, as illustrated in Figure 5.21.

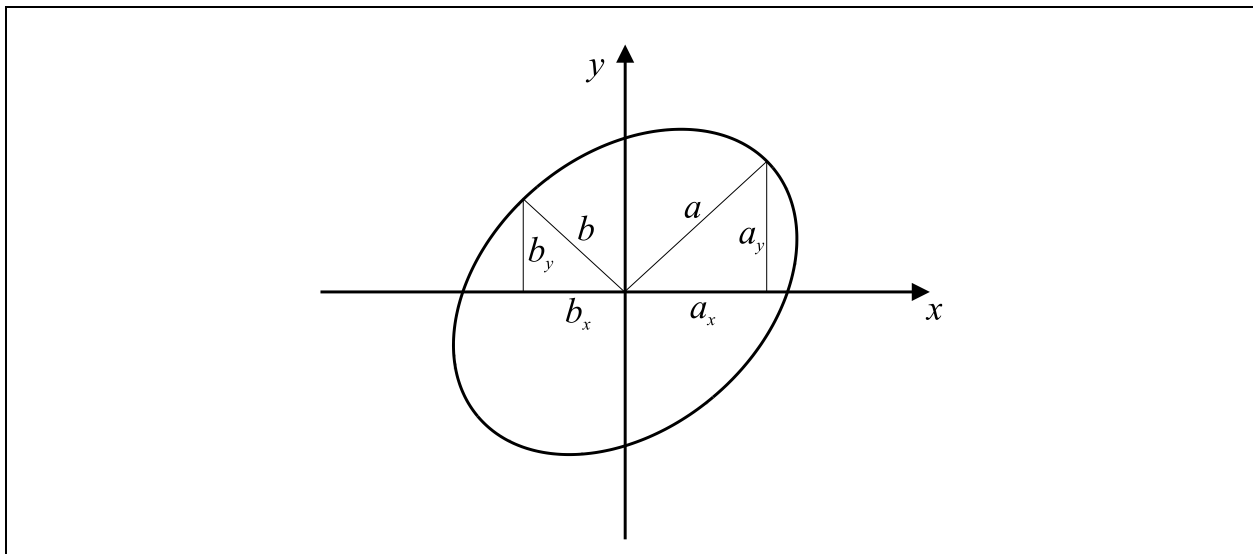


Figure 5.21 Definition of Ellipse Axes

In a similar way to Equation (5.33), Equation (5.36) can be used to generate the mapping function in the HT. In this case, the location of the centre of the ellipse is given by

$$\begin{aligned} a_0 &= x - a_x \cos(\theta) + b_x \sin(\theta) \\ b_0 &= y - a_y \cos(\theta) + b_y \sin(\theta) \end{aligned} \quad (5.39)$$

The location is dependent on three parameters, thus the mapping defines the trace of a hypersurface in a 5D space. This space can be very large. For example, if there are 100 possible values for each of the five parameters, the 5D accumulator space contains  $10^{10}$  values. This is 10 GB of storage, which is of course tiny nowadays (at least, when someone else pays!). Accordingly there has been much interest in ellipse detection techniques which use much less space and operate much faster than direct implementation of Equation (5.38).

Code 5.7 shows the implementation of the HT mapping for ellipses. A 5D accumulator imposes significant computational cost, thus the technique is only useful if it considers a small range for the parameters. In the code the parameters `majorAxisRange`, `minorAxisRange` and `angleRange` determine the possible values for the axis and rotation angles. The main loop in the function computes the centre parameters defined in Equation (5.39) for an ellipse according to rotation and axis lengths.

```
# Gather evidence
for x,y in itertools.product(range(0, width), range(0, height)):
    if magnitude[y,x] != 0:
        for majAxis, minAxis in itertools.product(range(0, majorAxisSize), \
                                                range(0, minorAxisSize)):

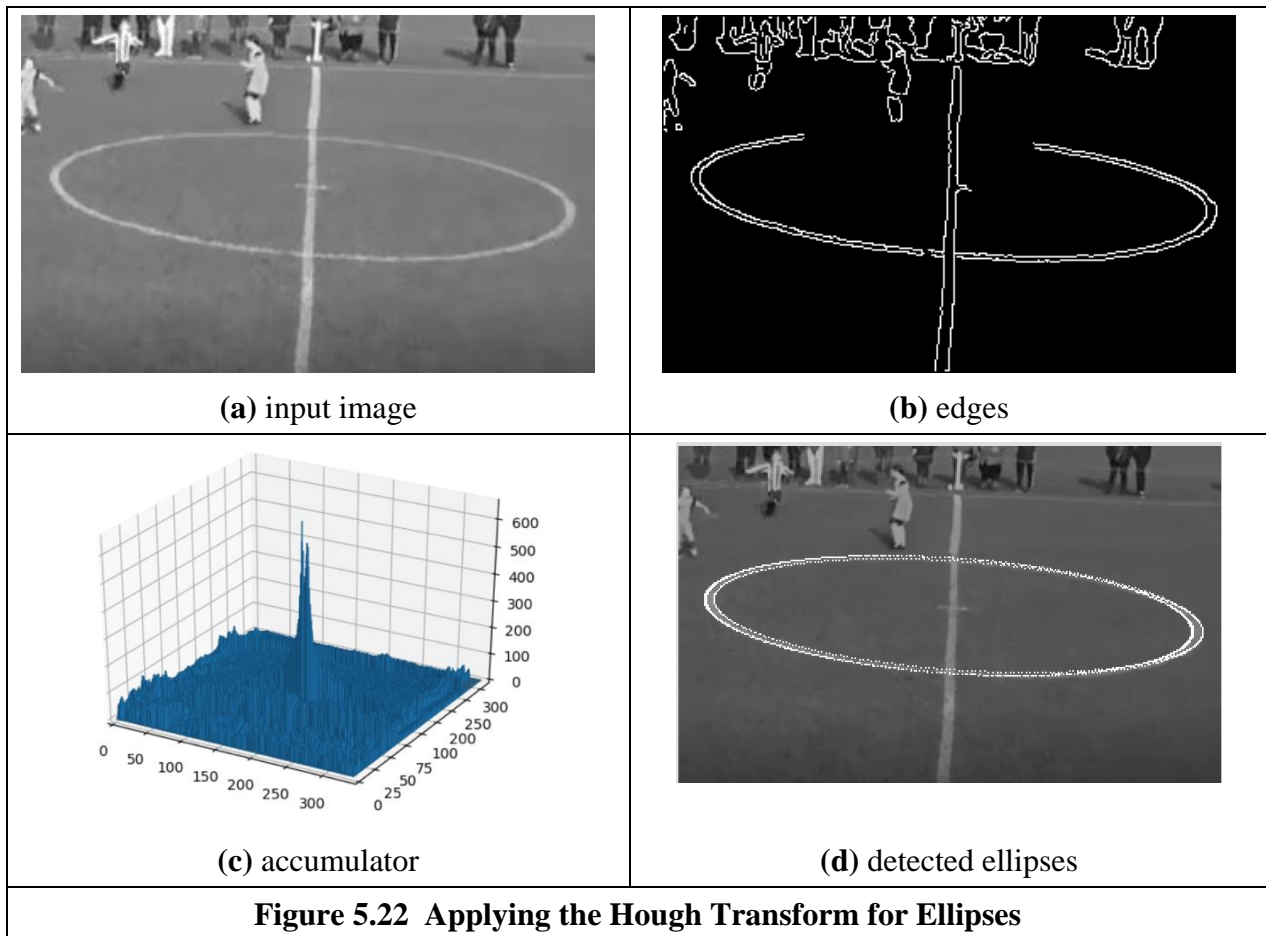
            a = majAxis + majorAxisRange[0]
            b = minAxis + minorAxisRange[0]
            for rot in range(0, angleSize):
                rotAngle = ((rot + angleRange[0]) * pi) / 180.0
                for m in range(0, 360):
                    angle = (m * pi) / 180.0

                    x0 = x+ a*cos(angle)*cos(rotAngle) - b*sin(angle)*sin(rotAngle)
                    y0 = y+ a*cos(angle)*sin(rotAngle) + b*sin(angle)*cos(rotAngle)
                    bx0 = int(x0)
                    by0 = int(y0)

                    if bx0>0 and bx0<width-1 and by0>0 and by0<height-1:
                        wx = x0 - bx0
                        wy = y0 - by0
                        accumulator[by0,bx0,majAxis,minAxis,rot] += (1.0-wX)+(1.0-wY)
                        accumulator[by0+1,bx0,majAxis,minAxis,rot] += wx + (1.0-wY)
                        accumulator[by0,bx0+1,majAxis,minAxis,rot] += (1.0-wX) + wY
                        accumulator[by0+1,bx0+1,majAxis,minAxis,rot] += wx + wY
```

### Code 5.7 Implementing the Hough Transform for Ellipses

Figure 5.22 shows an example of the application of the ellipse extraction process described in the Code 5.7. Figure 5.22(a) shows an image with an evident ellipse created from an oblique view of a circle. Figure 5.22(b) shows edges obtained with the Canny edge detector. A 2D slice of the 5D accumulator is shown in Figure 5.22(c). The array shows a peak whose position corresponds to the centre of the ellipse. We can observe that the accumulator shows close peaks near the ellipse location, so there is more than one ellipse to be located in the figure. The peaks are produced by the double edge of the ellipse so different possible ellipses can match parts of the internal or external edges. Figure 5.22(d) shows the ellipses that are obtained by considering the parameters defined by the maxima peaks. The rotation of all ellipses appears close to zero. However, the positions of different ellipses shift by a few pixels and the axis values vary slightly. This matches different segments of the interior and exterior ellipse's edges. Thus, as with the earlier examples for line and circle extraction, it is necessary to interpret the accumulator space in order to discover which structures produce particular parameter combinations.



### 5.5.5 Parameter Space Decomposition

The HT gives the same (optimal) result as template matching and even though it is faster, it still requires significant computational resources. In the previous sections, we saw that as we increase the complexity of the curve under detection, the computational requirements increase in exponential way. Thus, the HT becomes less practical. For this reason, most of the research in the HT has focused on the development of techniques aimed to reduce its computational complexity [Illingworth88] [Leavers93]. One important way to reduce the computation has been the use of geometric properties of shapes to decompose the parameter space. Several techniques have used different geometric properties. These geometric properties are generally defined by the relationship between points and derivatives.

#### 5.5.5.1 Parameter space reduction for lines

For a line, the accumulator space can be reduced from 2D to 1D by considering that we can compute the slope from the information of the image. The slope can be computed either by using the gradient direction at a point or by considering a pair of points. That is

$$m = \varphi \quad \text{or} \quad m = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.40}$$

where  $\varphi$  is the gradient direction at the point. In the case of take two points, by considering Equation (5.26) we have that,

$$c = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} \quad (5.41)$$

Thus, according to Equation (5.30) we have that one of the parameters of the polar representation for lines,  $\theta$ , is now given by

$$\theta = -\tan^{-1} \left[ \frac{1}{\phi} \right] \quad \text{or} \quad \theta = \tan^{-1} \left[ \frac{x_1 - x_2}{y_2 - y_1} \right] \quad (5.42)$$

These equations do not depend on the other parameter  $\rho$  and they provide alternative mappings to gather evidence. That is, they decompose the parametric space, such that the two parameters  $\theta$  and  $\rho$  are now independent. The use of edge direction information constitutes the base of the line extraction method presented by O'Gorman and Clowes [O'Gorman76]. The use of pairs of points can be related to the definition of the Randomised Hough Transform [Xu90]. Obviously, the number of feature points considered corresponds to all the combinations of points that form pairs. By using statistical techniques, it is possible to reduce the space of points in order to consider a representative sample of the elements. That is, a subset which provides enough information to obtain the parameters with predefined and small estimation errors.

Code 5.8 shows the implementation of the parameter space decomposition for the HT for lines. In this implementation, the slope of the lines is computed by considering a pair of points. Pairs of points are restricted to a neighbourhood given by the parameter `deltaPointRange`. This defines the minimum and maximum distance between points in a pair. In general, we do not want to have points that are close to each other since the gradient will not be accurate due to the image discretization. However, we do not want points too far apart since they may not belong to the same object in the image and they will only generate noise. As such, the range should be chosen by considering the noise, the image resolution and the size of the lines.

In Code 5.8 the implementation of Equation (5.42) is performed by using the function `atan2` that returns an angle between  $-180^\circ$  to  $180^\circ$ . Since our accumulators only can store positive values, then we add  $180^\circ$  to all values. The angle defined by two points is stored in the variable `pointAngle`. Notice that the centre point of the line (intersection of  $\rho$  with the line) can be out of the image, so in order to consider all possible lines we have to consider that  $\theta$  has a  $360^\circ$  range. The angle computed from the points is used, according to Equation (5.42), to compute  $\rho$ . Notice that to gather evidence we need to consider that the distance between points defines a discrete set of possible angles. For example, for points that are a single pixel apart there are only nine possible angles. Accordingly, the gathering process in the code increments the accumulator by a delta interval that is obtained according to the distance between the points.

```

# Gather evidence for a point x,y
for x,y in itertools.product(range(0, width), range(0, height)):
    if magnitude[y,x] != 0:
        # Look for points at this distance
        for dx,dy in itertools.product(range(-deltaPtRange[1],deltaPtRange[1]+1), \
                                       range(-deltaPtRange[1],deltaPtRange[1]+1)):

            if abs(dx) > deltaPtRange[0] or abs(dy) > deltaPtRange[0]:
                wx,wy = x+dx, y+dy
                if wx > 0 and wy > 0 and wx < width and wy < height \
                    and magnitude[wy, wx] !=0:
                    pointAngle = math.atan2(-float(wx-x), float(wy-y)) + math.pi

                    # If r is negative, the line is in the other side of the centre
                    r = (x-cx) * math.cos(pointAngle) + (y-cy) * math.sin(pointAngle)
                    if r < 0:
                        if pointAngle > math.pi: pointAngle -= math.pi
                        else: pointAngle += math.pi

                    # Accumulator entries depend on the distance to the second point
                    deltaDistance = math.sqrt(dx*dx + dy*dy)
                    incAngle = int(math.atan(1.0/deltaDistance) * 180.0 / math.pi)

                    bucketAngleBase = int((pointAngle * 180.0) / math.pi)

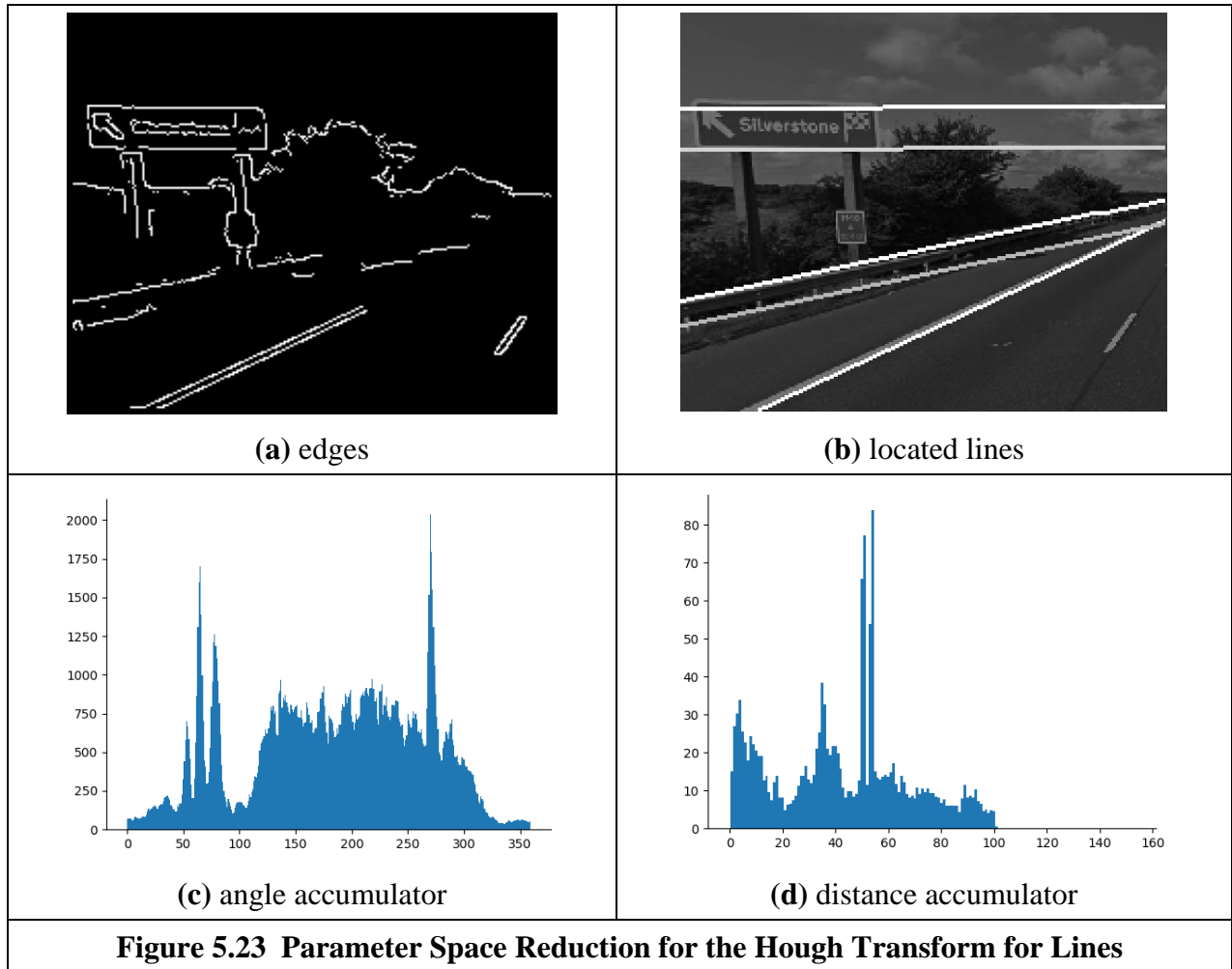
                    # More buckets if the points are close
                    for deltaBucket in range(-incAngle,+incAngle+1):
                        bucket = bucketAngleBase + deltaBucket
                        if bucket < 0:
                            bucket = 360 + bucket
                        if bucket >= 360:
                            bucket = bucket-360

                        w = (incAngle - math.fabs(deltaBucket)) / float(incAngle)
                        accM[bucket] += w

```

**Code 5.8 Implementation of the Parameter Space Reduction for the Hough Transform for Lines. Angle accumulator**

Figure 5.23 shows the accumulators for the two parameters  $\theta$  and  $\rho$  as obtained by the implementation of Code 5.8. The accumulators are now one dimensional as in Figure 5.23(c) and (d) and show clear peaks. The peaks in the first accumulator represent the value of  $\theta$ , whilst the peaks in the second accumulator represent the value of  $\rho$ . Notice that when implementing the parameter space decomposition, it is necessary to follow a two-step process. First, it is necessary to gather data in one accumulator and search for the maximum. Secondly, the location of the maximum value is used as parameter value to gather data of the remaining accumulator. Thus, we first compute the accumulator  $\theta$  and then for each peak (line), we generate an accumulator for  $\rho$ . That is, we have a single accumulator for the angle and as many accumulators for  $\rho$  as lines detected in the image. We can see that most of the peaks in the first accumulator are close to  $60^\circ$ , so with similar slopes. The peaks define the lines in Figure 5.23(b). It is important to mention that in general parameter space decomposition produces accumulators with more noise than when gathering evidence in the full dimensional accumulator. So it is important to include strategies to select good candidate points. In this implementation we considered pairs of points within a selected distance, but other selection criteria like points with similar gradient direction or similar colours can help to reduce the noise in the accumulators.



### 5.5.5.2 Parameter space reduction for circles

In the case of lines the relationship between local information computed from an image and the inclusion of a group of points (pairs) is in an alternative analytical description that can readily be established. For more complex primitives, it is possible to include several geometric relationships. These relationships are not defined for an arbitrary set of points but include angular constraints that define relative positions between them. In general, we can consider different geometric properties of the circle to decompose the parameter space. This has motivated the development of many methods of parameter space decomposition [Aguado96b]. An important geometric relationship is given by the geometry of the second directional derivatives. This relationship can be obtained by considering that Equation (5.33) defines a position vector function. That is,

$$v(\theta) = x(\theta) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y(\theta) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad (5.43)$$

where  $x(\theta) = x_0 + r \cos(\theta)$   $y(\theta) = y_0 + r \sin(\theta)$  (5.44)

In this definition, we have included the parameter of the curve as an argument in order to highlight the fact that the function defines a vector for each value of  $\theta$ . The end-points of all the vectors trace a circle. The derivatives of Equation (5.43) with respect to  $\theta$  define the first and second directional derivatives. That is,

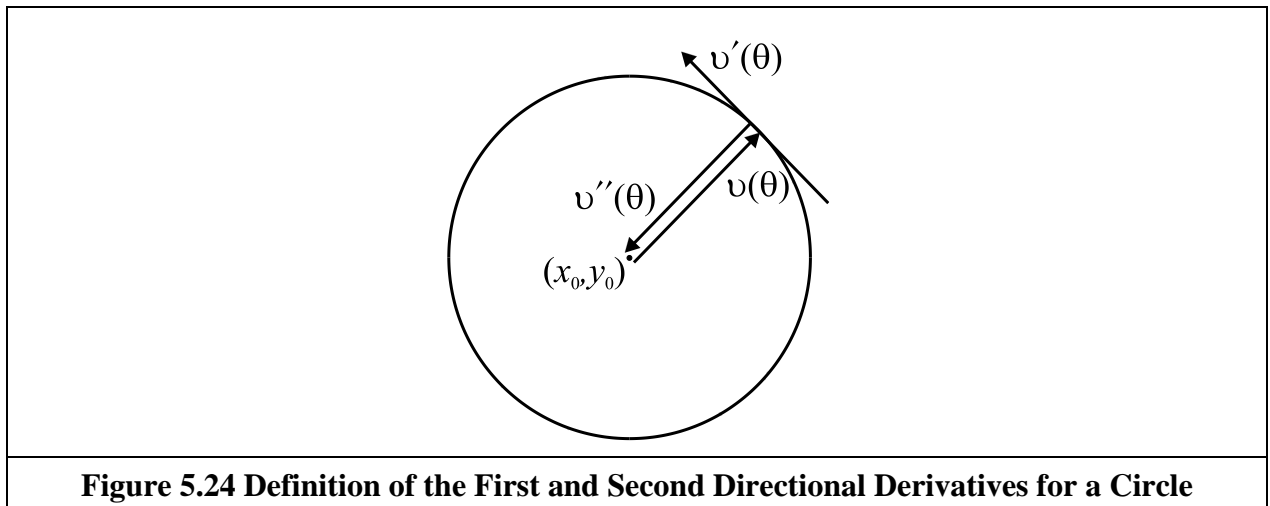


$$\begin{aligned} \mathbf{v}'(\theta) &= x'(\theta) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y'(\theta) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ \mathbf{v}''(\theta) &= x''(\theta) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y''(\theta) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned} \tag{5.45}$$

where

$$\begin{aligned} x'(\theta) &= -r \sin(\theta) & y'(\theta) &= r \cos(\theta) \\ x''(\theta) &= -r \cos(\theta) & y''(\theta) &= -r \sin(\theta) \end{aligned} \tag{5.46}$$

Figure 5.24 illustrates the definition of the first and second directional derivatives. The first derivative defines a tangential vector while the second one is similar to the vector function, but it has reverse direction. In fact, that the edge direction measured for circles can be arranged so as to point towards the centre was actually the basis of one of the early approaches to reducing the computational load of the HT for circles [Kimme75].



**Figure 5.24 Definition of the First and Second Directional Derivatives for a Circle**

According to Equation (5.44) and Equation (5.46), we observe that the tangent of the angle of the first directional derivative denoted as  $\hat{\phi}'(\theta)$  is given by

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} = -\frac{1}{\tan(\theta)} \tag{5.47}$$

Angles will be denoted by using the symbol  $\hat{\phi}$ . That is,

$$\hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \tag{5.48}$$

Similarly, for the tangent of the second directional derivative we have that,

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \tan(\theta) \quad \text{and} \quad \hat{\phi}''(\theta) = \tan^{-1}(\phi''(\theta)) \tag{5.49}$$

By observing the definition of  $\hat{\phi}''(\theta)$ , we have

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \frac{y(\theta) - y_0}{x(\theta) - x_0} \tag{5.50}$$

This equation defines a straight line passing through the points  $(x(\theta), y(\theta))$  and  $(x_0, y_0)$  and it is perhaps the most important relation in parameter space decomposition. The definition of the line is more evident by rearranging terms. That is,

$$y(\theta) = \phi''(\theta)(x(\theta) - x_0) + y_0 \tag{5.51}$$

This equation is independent of the radius parameter. Thus, it can be used to gather evidence of the location of the shape in a 2D accumulator. The HT mapping is defined by the dual form given by

$$y_0 = \phi''(\theta)(x_0 - x(\theta)) + y(\theta) \tag{5.52}$$

That is, given an image point  $(x(\theta), y(\theta))$  and the value of  $\phi''(\theta)$  we can generate a line of votes in the 2D accumulator  $(x_0, y_0)$ . Once the centre of the circle is known, then a 1D accumulator can be used to locate the radius. The key aspect of the parameter space decomposition is the method used to obtain the value of  $\phi''(\theta)$  from image data. We will consider two alternative ways. First, we will show that  $\phi''(\theta)$  can be obtained by edge direction information. Secondly, how it can be obtained from the information of a pair of points.

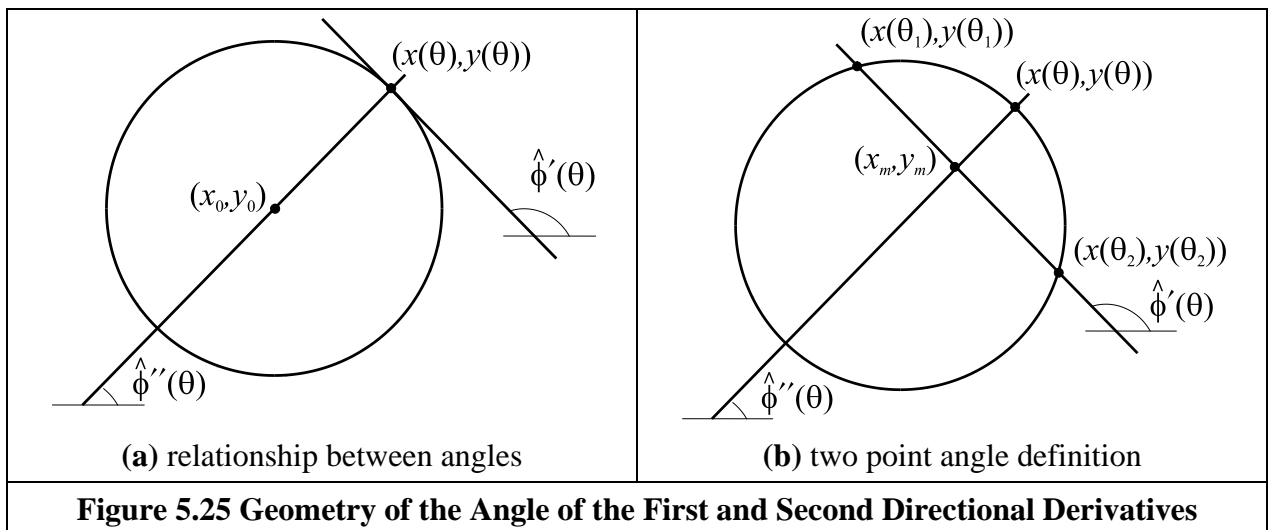
In order to obtain  $\phi''(\theta)$ , we can use the definition in Equation (5.47) and Equation (5.49). According to these equations, the tangents  $\phi''(\theta)$  and  $\phi'(\theta)$  are perpendicular. Thus,

$$\phi''(\theta) = -\frac{1}{\phi'(\theta)} \tag{5.53}$$

Thus, the HT mapping in Equation (5.52) can be written in terms of gradient direction  $\phi'(\theta)$  as

$$y_0 = y(\theta) + \frac{x(\theta) - x_0}{\phi'(\theta)} \tag{5.54}$$

This equation has a simple geometric interpretation illustrated in Figure 5.25(a). We can see that the line of votes passes through the points  $(x(\theta), y(\theta))$  and  $(x_0, y_0)$ . The slope of the line is perpendicular to the direction of gradient direction.



An alternative decomposition can be obtained by considering the geometry shown in Figure 5.25(b). In the figure we can see that if we take a pair of points  $(x_1, y_1)$  and  $(x_2, y_2)$ , where

$x_i = x(\theta_i)$ , then the line that passes through the points has the same slope as the line at a point  $(x(\theta), y(\theta))$ . Accordingly,

$$\phi'(\theta) = \frac{y_2 - y_1}{x_2 - x_1} \quad (5.55)$$

where 
$$\theta = \frac{1}{2}(\theta_1 + \theta_2) \quad (5.56)$$

Based on Equation (5.55) we have that

$$\phi''(\theta) = -\frac{x_2 - x_1}{y_2 - y_1} \quad (5.57)$$

The problem with using a pair of points is that by Equation (5.56) we cannot know the location of the point  $(x(\theta), y(\theta))$ . Fortunately, the voting line also passes through the midpoint of the line between the two selected points. Let us define this point as

$$x_m = \frac{1}{2}(x_1 + x_2) \quad y_m = \frac{1}{2}(y_1 + y_2) \quad (5.58)$$

Thus, by substitution of Equation (5.55) in (5.54) and by replacing the point  $(x(\theta), y(\theta))$  by  $(x_m, y_m)$ , we have that the HT mapping can be expressed as

$$y_0 = y_m + \frac{(x_m - x_0)(x_2 - x_1)}{(y_2 - y_1)} \quad (5.59)$$

This equation does not use gradient direction information, but it is based on pairs of points. This is analogous to the parameter space decomposition of the line presented in Equation (5.42). In that case, the slope can be computed by using gradient direction or, alternatively, by taking a pair of points. In the case of the circle, the tangent (and therefore the angle of the second directional derivative) can be computed by the gradient direction (i.e., Equation 5.53) or by a pair of points (i.e., Equation 5.57). However, it is important to notice that there are some other combinations of parameter space decomposition [Aguado96a].

```
# Gather evidence for the circle location by using two points
accumulator = createImageF(width, height)
for x1,y1 in itertools.product(range(0, width), range(0, height)):
    if magnitude[y1,x1] != 0:
        # Look for points at this distance
        for dx,dy in itertools.product(range(0,deltaPointRange[1]+1),
                                       range(0,deltaPointRange[1]+1)):
            if (dx!=0 or dy!=0) and (abs(dx) > deltaPointRange[0] or
                                     abs(dy) > deltaPointRange[0]):
                x2, y2 = x1+dx, y1+dy
                if x2 > 0 and y2 > 0 and x2 < width and y2 < height and
                    magnitude[y2, x2] !=0:
                    xm, ym = (x1 + x2) / 2.0, (y1 + y2) / 2.0
                    if abs(dx) < abs(dy):
                        m = float(dx) / float(-dy)
                        for x in range(0, width):
                            y = m * (x - xm) + ym
                            intY = int(y)
                            if intY > 0 and intY < height -1:
                                weight = y - intY
                                accumulator[ intY, x] += (1.0 - weight)
                                accumulator[ intY+1, x] += weight
                    else:
                        m = float(-dy) / float(dx)
                        for y in range(0, height):
                            x = m * (y - ym) + xm
```

```

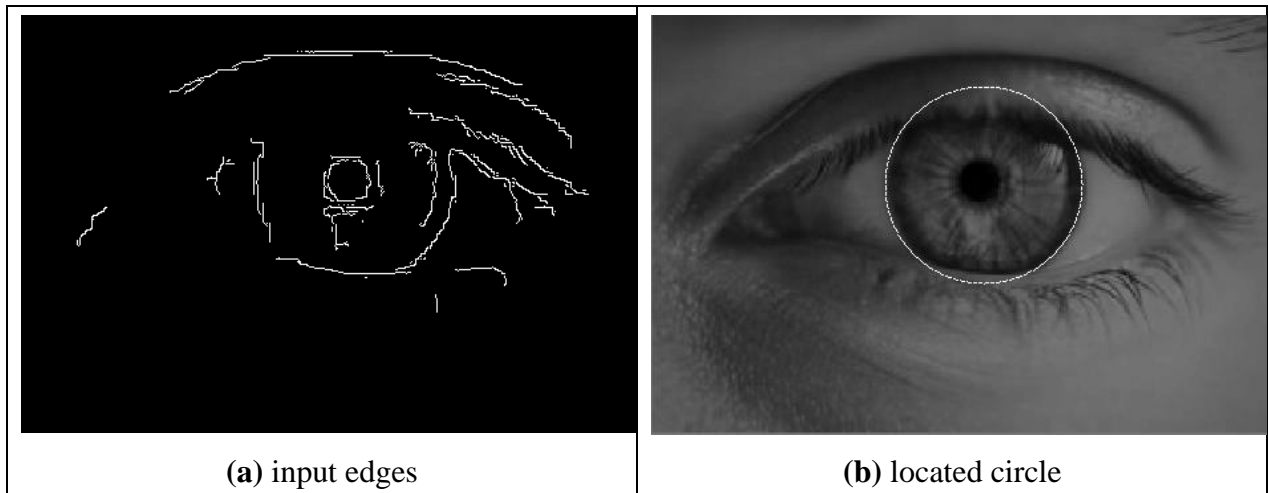
intX = int(x)
if intX > 0 and intX < width -1:
    weight = x - intX
    accumulator[ y, intX] += (1.0 - weight)
    accumulator[ y, intX+1] += weight

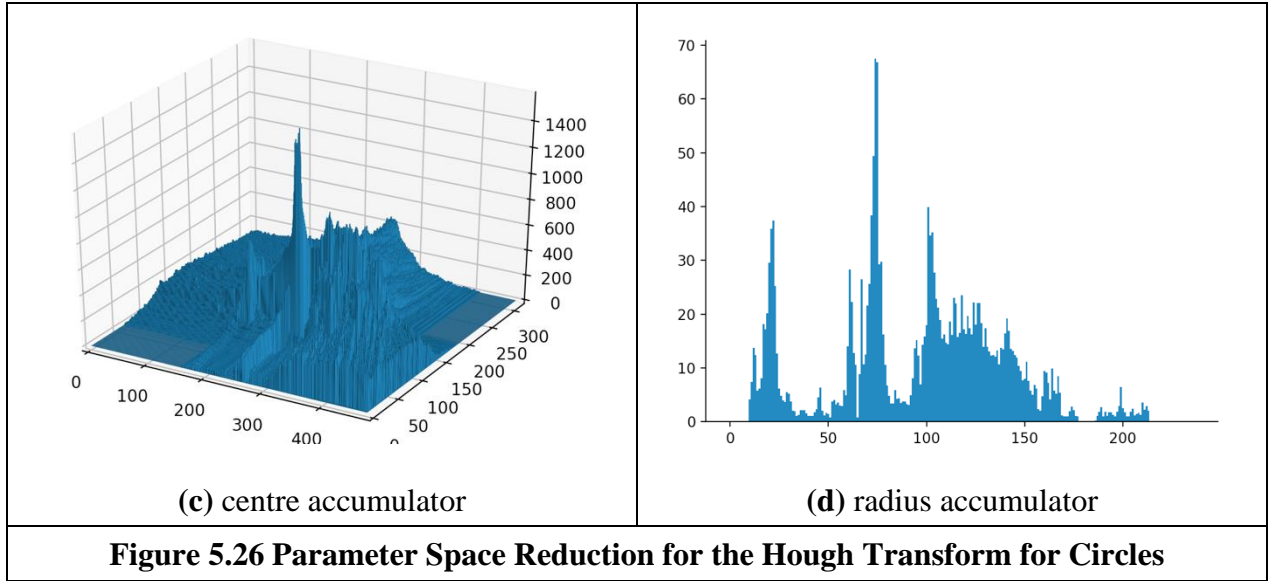
```

**Code 5.9 Parameter Space Reduction for the Hough Transform for Circles**

Code 5.9 shows the implementation of the parameter space decomposition for the HT for circles. The implementation only detects the position of the circle and it gathers evidence by using the mapping in Equation (5.59). Similar to the implementation of lines, in this case points are selected from a neighbourhood defined by the parameter `deltaPointRange`. As such, we avoid using pixels that are close to each other since they do not produce accurate votes and we also avoid using pixels that are far away from each other. In order to trace the line in Equation (5.59), we use two equations: one for horizontal and one for vertical lines. As in previous implementations, it is important to include weighting to reduce discretization errors.

Figure 5.26 shows an example of the results obtained by the implementation in Code 5.9. The example shows the input edges obtained by the Canny edge detector for the eye image in Figure 5.20. We can see that both accumulators show a clear peak that represents the location of the circle. Small peaks in the background of the accumulator in Figure 5.26(b) correspond to circles with only a few points. In general, there is a compromise between the width of the peak and the noise in the accumulator. The peak can be made narrower by considering pairs of points that are more widely spaced. However, this can also increase the level of background noise. Background noise can be reduced by taking points that are closer together, but this makes the peak wider. The accumulator in Figure 5.26(d) shows the evidence gathered for the circle radius. This accumulator is obtained by selecting the location parameters as the maximum in the accumulator in Figure 5.26(c) and then computing the gather evidence for the radius for each image point according to Equation 5.32.





### 5.5.5.3 Parameter space reduction for ellipses

Part of the simplicity in the parameter decomposition for circles comes from the fact that circles are (naturally) isotropic. Ellipses have more free parameters and are geometrically more complex. Thus, geometrical properties involve more complex relationships between points, tangents and angles. However, they maintain the geometric relationship defined by the angle of the second derivative. According to Equation (5.43) and Equation (5.45), the vector position and directional derivatives of an ellipse in Equation (5.37) have the components

$$\begin{aligned} x'(\theta) &= -a_x \sin(\theta) + b_x \cos(\theta) & y'(\theta) &= -a_y \sin(\theta) + b_y \cos(\theta) \\ x''(\theta) &= -a_x \cos(\theta) - b_x \sin(\theta) & y''(\theta) &= -a_y \cos(\theta) - b_y \sin(\theta) \end{aligned} \quad (5.60)$$

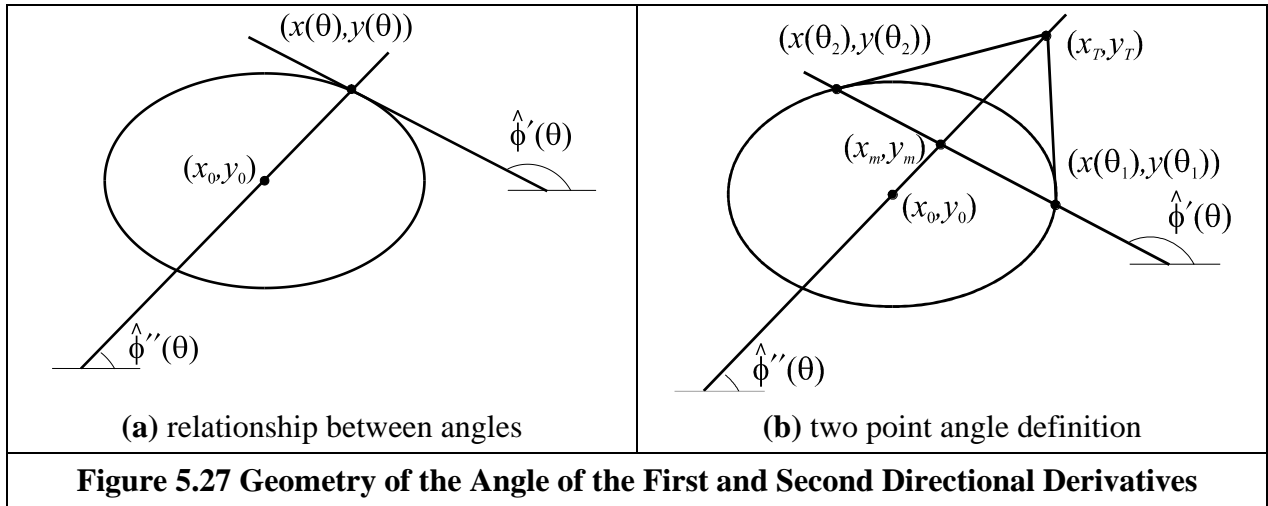
The tangent of angle of the first and second directional derivatives are given by

$$\begin{aligned} \phi'(\theta) &= \frac{y'(\theta)}{x'(\theta)} = \frac{-a_y \sin(\theta) + b_y \cos(\theta)}{-a_x \sin(\theta) + b_x \cos(\theta)} \\ \phi''(\theta) &= \frac{y''(\theta)}{x''(\theta)} = \frac{-a_y \cos(\theta) - b_y \sin(\theta)}{-a_x \cos(\theta) - b_x \sin(\theta)} \end{aligned} \quad (5.61)$$

By considering Equation (5.59) we have that Equation (5.49) is also valid for an ellipse. That is,

$$\frac{y(\theta) - y_0}{x(\theta) - x_0} = \phi''(\theta) \quad (5.62)$$

The geometry of the definition in this equation is illustrated in Figure 5.27 (a). As in the case of circles, this equation defines a line that passes through the points  $(x(\theta), y(\theta))$  and  $(x_0, y_0)$ . However, in the case of the ellipse the angles  $\hat{\phi}'(\theta)$  and  $\hat{\phi}''(\theta)$  are not orthogonal. This makes the computation of  $\phi''(\theta)$  more complex. In order to obtain  $\phi''(\theta)$  we can extend the geometry presented in Figure 5.25(b). That is, we take a pair of points to define a line whose slope defines the value of  $\phi'(\theta)$  at another point. This is illustrated in Figure 5.27(b). The line in Equation (5.62) passes through the middle point  $(x_m, y_m)$ . However, it is not orthogonal to the tangent line. In order to obtain an expression of the HT mapping, we will first show that the relationship in Equation (5.56) is also valid for ellipses. Then we will use this equation to obtain  $\phi''(\theta)$ .



The relationships in Figure 5.27(b) do not depend on the orientation or position of the ellipse. Thus, we have that three points can be defined by

$$\begin{aligned} x_1 &= a_x \cos(\theta_1) & x_2 &= a_x \cos(\theta_2) & x(\theta) &= a_x \cos(\theta) \\ y_1 &= b_y \sin(\theta_1) & y_2 &= b_y \sin(\theta_2) & y(\theta) &= b_y \sin(\theta) \end{aligned} \tag{5.63}$$

The point  $(x(\theta), y(\theta))$  is given by the intersection of the line in Equation (5.62) with the ellipse. That is,

$$\frac{y(\theta) - y_0}{x(\theta) - x_0} = \frac{a_x}{b_y} \frac{y_m}{x_m} \tag{5.64}$$

By substitution of the values of  $(x_m, y_m)$  defined as the average of the co-ordinates of the points  $(x_1, y_1)$  and  $(x_2, y_2)$  in Equation (5.58), we have that

$$\tan(\theta) = \frac{a_x}{b_y} \frac{b_y \sin(\theta_1) + b_y \sin(\theta_2)}{a_x \cos(\theta_1) + a_x \cos(\theta_2)} \tag{5.65}$$

Thus, 
$$\tan(\theta) = \tan\left(\frac{1}{2}(\theta_1 + \theta_2)\right) \tag{5.66}$$

From this equation is evident that the relationship in Equation (5.56) is also valid for ellipses. Based on this result, the tangent angle of the second directional derivative can be defined as

$$\phi''(\theta) = \frac{b_y}{a_x} \tan(\theta) \tag{5.67}$$

By substitution in Equation (5.64) we have that

$$\phi''(\theta) = \frac{y_m}{x_m} \tag{5.68}$$

This equation is valid when the ellipse is not translated. If the ellipse is translated then the tangent of the angle can be written in terms of the points  $(x_m, y_m)$  and  $(x_T, y_T)$  as

$$\phi''(\theta) = \frac{y_T - y_m}{x_T - x_m} \tag{5.69}$$

By considering that the point  $(x_T, y_T)$  is the intersection point of the tangent lines at  $(x_1, y_1)$  and  $(x_2, y_2)$  we obtain

$$\phi''(\theta) = \frac{AC + 2BD}{2A + BC} \quad (5.70)$$

where

$$\begin{aligned} A &= y_1 - y_2 & B &= x_1 - x_2 \\ C &= \phi_1 + \phi_2 & D &= \phi_1 \cdot \phi_2 \end{aligned} \quad (5.71)$$

and  $\phi_1, \phi_2$  are the slope of the tangent line to the points. Finally, by considering Equation (5.62), the HT mapping for the centre parameter is defined as

$$y_0 = y_m + \frac{AC + 2BD}{2A + BC}(x_0 - x_m) \quad (5.72)$$

This equation can be used to gather evidence that is independent of rotation or scale. Once the location is known, it is necessary a 3D parameter space to obtain the remaining parameters. However, these parameters can also be computed independently using two 2D parameter spaces [Aguado96b]. Of course that you can avoid using the gradient direction in Equation (5.70) by including more points. In fact, the tangent  $\phi''(\theta)$  can be computed by taking four points [Aguado96a]. However, the inclusion of more points generally leads to more background noise in the accumulator.

Code 5.10 shows the implementation of the ellipse location by the mapping in Equation (5.59). As in the case of the circle, pairs of points need to be restricted. However, in this implementation we show a different approach to select pair of points. Instead of simply consider a distance range, we create curve segments by considering the neighbour pixels. In general, we expect that pixels in segments to belong to the same primitive, so pair of pixels should provide better evidence than only selecting pixels in an image region. The segments are stored in the `segments` array. Pairs of points are formed by selecting random points in the segments. The implementation rejects pairs of points that have similar gradients since they produce a polar point that tends to infinity. Consequently, points with similar slope do not provide accurate evidence. We should notice that as the eccentricity of the ellipse increases, close points tend to have similar gradients making the detection process more difficult. Also notice that the gradient is noisy and not very accurate, so the resulting peak is generally quite wide. Again, the selection of the distance between points is a compromise between the level of background noise and the width of the peak. A relevant aspect of the implementation is that it is necessary to consider that evidence is actually gathered by tracing a line in the accumulator. Thus, as in the previous implementation, it is important to consider the slope of the line during the accumulation process. For horizontal lines  $x$  is the independent variable whilst for vertical lines we use  $y$ .

Figure 5.28 shows an example of the accumulators obtained by the implementation of Code 5.10. The peak in Figure 5.28(c) represents the location of the ellipses. In general, there is noise and the accumulator peak is spread out. This is for two main reasons. First, when the gradient direction is not accurate, then the line of votes does not pass precisely over the centre of the ellipse. This forces the peak to become wider with less height. Secondly, in order to avoid numerical instabilities, we need to select points that are well separated. However, this increases the probability that the points do not belong to the same ellipse, thus generating background noise in the accumulator. The accumulator in Figure Figure 5.28(d) shows the accumulator for the axis and rotation parameters. The figure shows a 2D slice of the 3D accumulator at the maximum point.

```

# Gather evidence for the ellipse location on the valid points
accumulator = createImageF(width, height)
numPoints = len(segments)

# For a pair p1 = (x1,y1), p2=(x2,y2)
for p1 in range(0, numPoints):
    for p in range(0,pairsPerPoint):
        p2 = randint(0, numPoints-1)

        y1,x1 = (segments[p1])[0], (segments[p1])[1]
        y2,x2 = (segments[p2])[0], (segments[p2])[1]
        d = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2))
        if d > deltaPointRange[0]:
            angle1, angle2 = -angle[y1,x1], -angle[y2,x2]

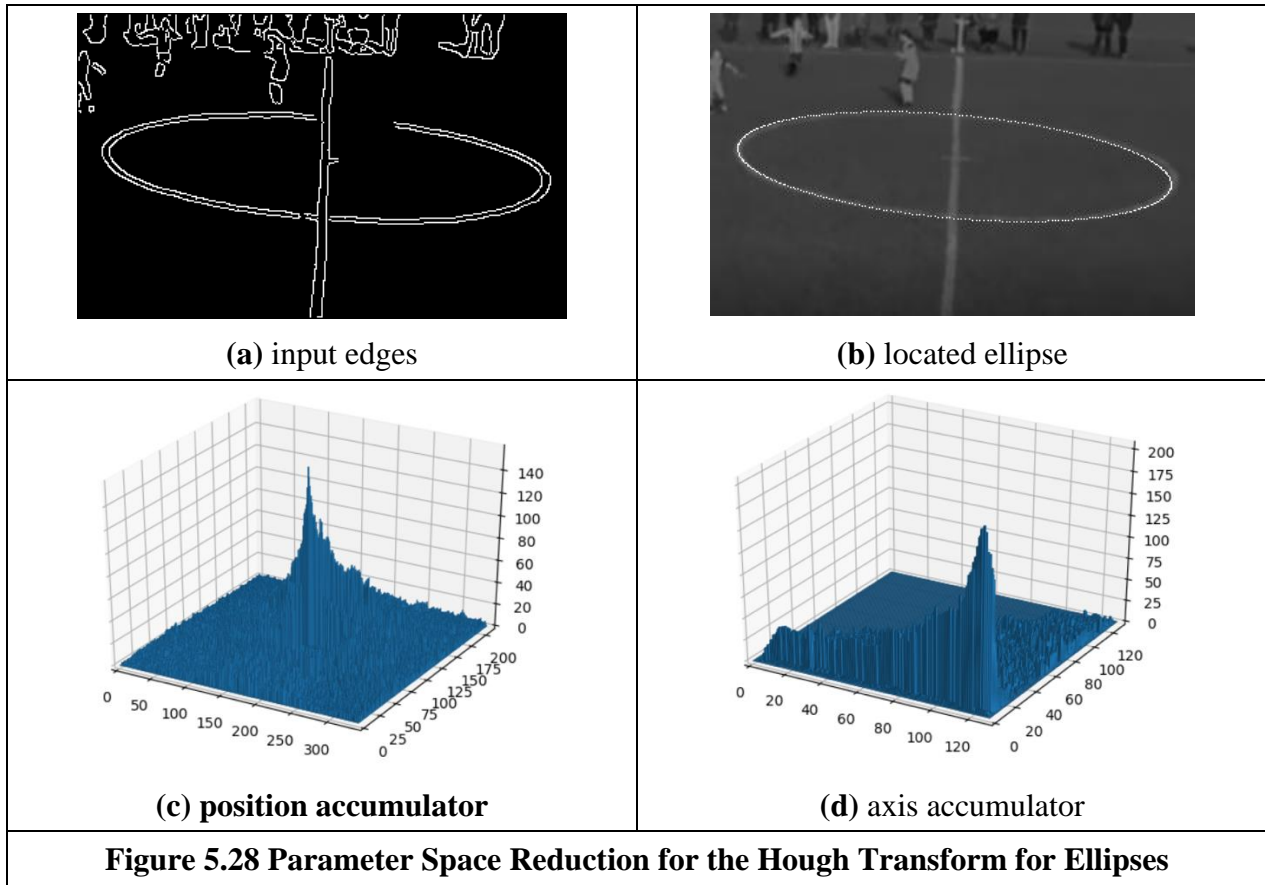
            # To void parallel edge directions
            w = cos(angle1)*cos(angle2) + sin(angle1)*sin(angle2)
            if w < 0.9:
                xm, ym = (x1 + x2) / 2.0, (y1 + y2) / 2.0
                m1, m2 = tan(angle1), tan(angle2)
                A,B = y1-y2, x2-x1
                C,D = m1+m2, m1*m2
                M,N = A*C+2*B*D, 2*A+B*C

                norm = sqrt(M*M+N*N)
                M,N = M/norm, N/norm
                # Draw horizontal or vertical lines
                if abs(M) < abs(N):
                    m = float(M) / float(N)
                    b1, b2 = y1-m1*x1, y2-m2*x2
                    xIntersect = (b2-b1) / (m1-m2)
                    if xIntersect < xm:
                        xi,xf = int(xm), min(int(xm + axisRange[1]), width-1)
                    else:
                        xi,xf = max(1,int(xm - axisRange[1])), int(xm)
                    for x in range(xi,xf):
                        y = m * (x - xm) + ym
                        dl = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1))
                        if dl > axisRange[0] and dl < axisRange[1]:
                            yInt = int(y)
                            if yInt > 0 and yInt < height -1:
                                weight = y - yInt
                                accumulator[yInt, x] += (1.0 - weight)
                                accumulator[yInt+1, x] += weight
                else:
                    m = float(N) / float(M)
                    b1, b2 = x1-m1*y1, x2-m2*y2
                    yIntersect = (b2-b1) / (m1-m2)
                    if yIntersect < ym:
                        yi,yf = int(ym), min(int(ym + axisRange[1]), height-1)
                    else:
                        yi,yf = max(1,int(ym - axisRange[1])), int(ym)
                    for y in range(yi,yf):
                        x = m * (y - ym) + xm
                        dl = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1))
                        if dl > axisRange[0] and dl < axisRange[1]:
                            xInt = int(x)
                            if xInt > 0 and xInt < width -1:
                                weight = x - xInt
                                accumulator[y, xInt] += (1.0 - weight)
                                accumulator[y, xInt+1] += weight

```

**Code 5.10 Implementation of the Parameter Space Reduction for the Hough Transform for Ellipses**





### 5.5.6 Generalised Hough Transform (GHT)

Many shapes are far more complex than lines, circles or ellipses. It is often possible to partition a complex shape into several geometric primitives, but this can lead to a highly complex data structure. In general, it is more convenient to extract the whole shape. This has motivated the development of techniques that can find arbitrary shapes using the evidence-gathering procedure of the HT. These techniques again give results equivalent to those delivered by matched template filtering, but with the computational advantage of the evidence gathering approach. An early approach offered only limited capability only for arbitrary shapes [Merlin75]. The full mapping is called the *Generalised HT* (GHT) [Ballard81] and can be used to locate arbitrary shapes with unknown position, size and orientation. The GHT can be formally defined by considering the duality of a curve. One possible implementation can be based on the discrete representation given by tabular functions. These two aspects are explained in the following two sections.

#### 5.5.6.1 Formal Definition of the GHT

The formal analysis of the HT provides the route for generalising it to arbitrary shapes. We can start by generalising the definitions in Equation (5.43). In this way a model shape can be defined by a curve

$$v(\theta) = x(\theta) \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y(\theta) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{5.73}$$

For a circle, for example, we have that  $x(\theta) = r \cos(\theta)$  and  $y(\theta) = r \sin(\theta)$ . Any shape can be represented by following a more complex definition of  $x(\theta)$  and  $y(\theta)$ .

In general, we are interested in matching the model shape against a shape in an image. However, the shape in the image has a different location, orientation and scale. Originally the GHT defined a scale parameter in the  $x$  and  $y$  directions, but due to computational complexity and practical relevance the use of a single scale has become much more popular. Analogous to Equation (5.35), we can define the image shape by considering translation, rotation and change of scale. Thus, the shape in the image can be defined as

$$\omega(\theta, b, \lambda, \rho) = b + \lambda \mathbf{R}(\rho) \upsilon(\theta) \quad (5.74)$$

where  $b = (x_0, y_0)$  is the translation vector,  $\lambda$  is a scale factor and  $\mathbf{R}(\rho)$  is a rotation matrix (as in Equation (5.33)). Here we have included explicitly the parameters of the transformation as arguments, but to simplify the notation they will be omitted later. The shape of  $\omega(\theta, b, \lambda, \rho)$  depends on four parameters. Two parameters define the location  $b$ , plus the rotation and scale. It is important to notice that  $\theta$  does not define a free parameter, and only traces the curve.

In order to define a mapping for the HT we can follow the approach used to obtain Equation (5.37). Thus, the location of the shape is given by

$$b = \omega(\theta) - \lambda \mathbf{R}(\rho) \upsilon(\theta) \quad (5.75)$$

Given a shape  $\omega(\theta)$  and a set of parameters  $b$ ,  $\lambda$  and  $\rho$ , this equation defines the location of the shape. However, we do not know the shape  $\omega(\theta)$  (since it depends on the parameters that we are looking for), but we only have a point in the curve. If we call  $\omega_i = (\omega_{xi}, \omega_{yi})$  the point in the image, then

$$b = \omega_i - \lambda \mathbf{R}(\rho) \upsilon(\theta) \quad (5.76)$$

defines a system with four unknowns and with as many equations as points in the image. In order to find the solution we can gather evidence by using a four dimensional accumulator space. For each potential value of  $b$ ,  $\lambda$  and  $\rho$ , we trace a point spread function by considering all the values of  $\theta$ . That is, all the points in the curve  $\upsilon(\theta)$ .

In the GHT the gathering process is performed by adding an extra constraint to the system that allows us to match points in the image with points in the model shape. This constraint is based on gradient direction information and can be explained as follows. We said that ideally, we would like to use Equation (5.75) to gather evidence. For that we need to know the shape  $\omega(\theta)$  and the model  $\upsilon(\theta)$ , but we only know the discrete points  $\omega_i$  and we have supposed that these are the same as the shape, i.e. that  $\omega(\theta) = \omega_i$ . Based on this assumption, we then consider all the potential points in the model shape,  $\upsilon(\theta)$ . However, this is not necessary since we only need the point in the model,  $\upsilon(\theta)$ , that corresponds to the point in the shape,  $\omega(\theta)$ . We cannot know the point in the shape,  $\upsilon(\theta)$ , but we can compute some properties from the model and from the image. Then, we can check whether these properties are similar at the point in the model and at a point in the image. If they are indeed similar, the points might correspond: if they do we can gather evidence of the parameters of the shape. The GHT considers the gradient direction at the point as a feature. We can generalise Equation (5.47) and Equation (5.48) to define the gradient direction at a point in the arbitrary model. Thus,

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} \quad \text{and} \quad \hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \quad (5.77)$$

Thus Equation (5.75) is true only if the gradient direction at a point in the image matches the rotated gradient direction at a point in the (rotated) model, that is

$$\phi'_i = \hat{\phi}'(\theta) - \rho \quad (5.78)$$

where  $\hat{\phi}'(\theta)$  is the angle at the point  $\omega_i$ . Note that according to this equation, gradient direction is independent of scale (in theory at least) and it changes in the same ratio as rotation. We can constrain Equation (5.76) to consider only the points  $\upsilon(\theta)$  for which

$$\phi'_i - \hat{\phi}'(\theta) + \rho = 0 \quad (5.79)$$

That is, a point spread function for a given edge point  $\omega_i$  is obtained by selecting a subset of points in  $\upsilon(\theta)$  such that the edge direction at the image point rotated by  $\rho$  equals the gradient direction at the model point. For each point  $\omega_i$  and selected point in  $\upsilon(\theta)$  the point spread function is defined by the HT mapping in Equation (5.76).

### 5.5.6.2 Polar definition

Equation (5.76) defines the mapping of the HT in Cartesian form. That is, it defines the votes in the parameter space as a pair of co-ordinates  $(x, y)$ . There is an alternative definition in polar form. The polar implementation is more common than the Cartesian form [Hecker94][Sonka94]. The advantage of the polar form is that it is easy to implement since changes in rotation and scale correspond to addition in the angle-magnitude representation. However, ensuring that the polar vector has the correct direction incurs more complexity.

Equation (5.76) can be written in a form that combines rotation and scale as

$$b = \omega(\theta) - \gamma(\lambda, \rho) \quad (5.80)$$

where  $\gamma^T(\lambda, \rho) = [\gamma_x(\lambda, \rho) \ \gamma_y(\lambda, \rho)]$  and where the combined rotation and scale is

$$\begin{aligned} \gamma_x(\lambda, \rho) &= \lambda(x(\theta)\cos(\rho) - y(\theta)\sin(\rho)) \\ \gamma_y(\lambda, \rho) &= \lambda(x(\theta)\sin(\rho) + y(\theta)\cos(\rho)) \end{aligned} \quad (5.81)$$

This combination of rotation and scale defines a vector,  $\gamma(\lambda, \rho)$ , whose tangent angle and magnitude are given by

$$\tan(\alpha) = \frac{\gamma_y(\lambda, \rho)}{\gamma_x(\lambda, \rho)} \quad r = \sqrt{\gamma_x^2(\lambda, \rho) + \gamma_y^2(\lambda, \rho)} \quad (5.82)$$

The main idea here is that if we know the values for  $\alpha$  and  $r$ , then we can gather evidence by considering Equation (5.80) in polar form. That is,

$$b = \omega(\theta) - r e^{j\alpha} \quad (5.83)$$

Thus, we should focus on computing values for  $\alpha$  and  $r$ . After some algebraic manipulation, we have that

$$\alpha = \phi(\theta) + \rho \quad r = \lambda \Gamma(\theta) \quad (5.84)$$

where

$$\phi(\theta) = \tan^{-1}\left(\frac{y(\theta)}{x(\theta)}\right) \quad \Gamma(\theta) = \sqrt{x^2(\theta) + y^2(\theta)} \quad (5.85)$$

In this definition, we must include the constraint defined in Equation (5.79). That is, we gather evidence only when the gradient direction is the same. Notice that the square root in the definition of the magnitude in Equation (5.85) can have positive and negative values. The sign must be selected in a way that the vector has the correct direction.

### 5.5.6.3 The GHT Technique

Equations (5.76) and (5.83) define an HT mapping function for arbitrary shapes. The geometry of these equations is shown in Figure 5.29. Given an image point  $\omega_i$  we have to find a displacement vector  $\gamma(\lambda, \rho)$ . When the vector is placed at  $\omega_i$ , then its end is at the point  $b$ . In the GHT jargon, this point called the reference point. The vector  $\gamma(\lambda, \rho)$  can be easily obtained as  $\lambda R(\rho) \upsilon(\theta)$  or alternative as  $re^\alpha$ . However, in order to evaluate these equations, we need to know the point  $\upsilon(\theta)$ . This is the crucial step in the evidence gathering process. Notice the interesting similarity between Figure 5.25(a), Figure 5.27(a) and Figure 5.29(a). This is not a coincidence, but Equation (5.62) is a particular case of Equation (5.75).

The process of determining  $\upsilon(\theta)$  centres on solving Equation (5.78). According to this equation, since we know  $\hat{\phi}'_i$ , then we need to find the point  $\upsilon(\theta)$  whose gradient direction is  $\hat{\phi}'_i + \rho = 0$ . Then we must use  $\upsilon(\theta)$  to obtain the displacement vector  $\gamma(\lambda, \rho)$ . The GHT pre-computes the solution of this problem and stores it in an array called the *R-table*. The *R-table* stores for each value of  $\hat{\phi}'_i$  the vector  $\gamma(\lambda, \rho)$  for  $\rho = 0$  and  $\lambda = 1$ . In polar form, the vectors are stored as a magnitude direction pair and in Cartesian form as a co-ordinate pair.

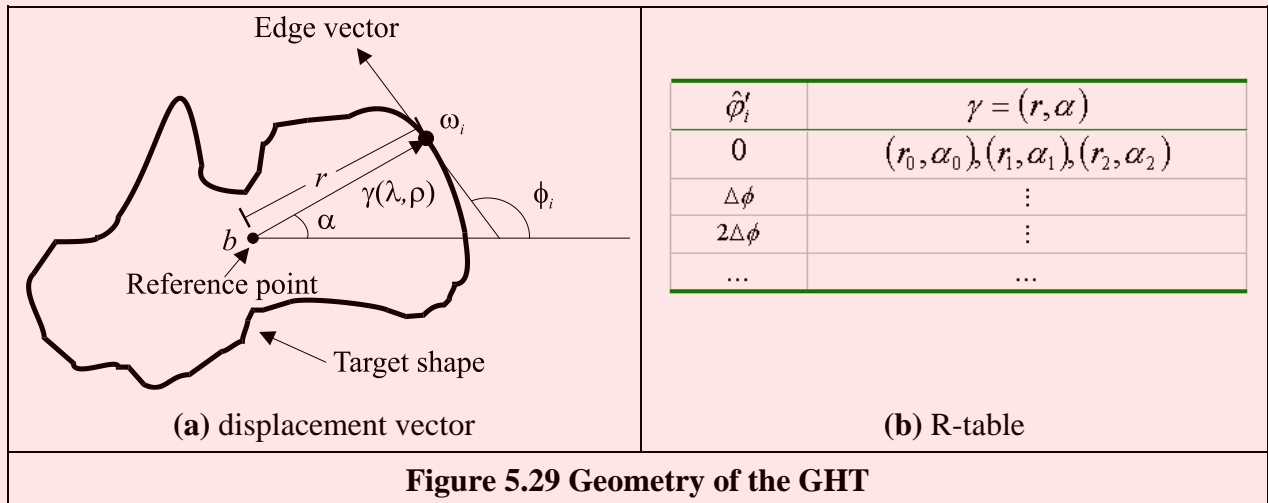


Figure 5.29 Geometry of the GHT

The possible range for  $\hat{\phi}'_i$  is between  $-\pi/2$  and  $\pi/2$  radians. This range is split into  $N$  equispaced slots, or bins. These slots become rows of data in the *R-table*. The edge direction at each border point determines the appropriate row in the *R-table*. The length,  $r$ , and direction,  $\alpha$ , from the reference point is entered into a new column element, at that row, for each border point in the shape. In this manner, the  $N$  rows of the *R-table* have elements related to the border information, elements for which there is no information contain null vectors. The length of each row is given by the number of edge points that have the edge direction corresponding to that row; the total number of

elements in the R-table equals the number of edge points above a chosen threshold. The structure of the R-table for  $N$  edge direction bins and  $m$  template border points is illustrated in Figure 5.29(b).

The process of building the R-table is illustrated in Code 5.11. In this code, we implement the Cartesian definition given in Equation (5.76). According to this equation the displacement vector is given by

$$\gamma(r, \alpha) = w(\theta) + b \quad (5.86)$$

The code stores the reference point  $b$  in the variable `refPoint`. The edge points are stored in the `edgePoints` list that is created by iterating over all the pixels in the template image. The list `rTable` is built for the number of entries defined by the parameter `numEntries`. Each entry covers an angle range defined in the variable `deltaAngle`. Each edge point is added as an entry in the table according to the angle stored in `angleTemplate`. This angle is the direction of the edge and was obtained when edges are detected by the Canny operator.

```
# Compute reference point in the template
refPoint = [0,0]
edgePoints = []
for x,y in itertools.product(range(0, widthTemplate), range(0, heightTemplate)):
    if magnitudeTemplate[y,x] != 0:
        refPoint[0] += y
        refPoint[1] += x
        edgePoints.append((y,x))
numPts = len(edgePoints)
refPoint = [int(refPoint[0]/numPts),int(refPoint[1]/numPts)]

# Build Rtable as a list of lists
rTable = [[] for entryIndex in range(numEntries)]
deltaAngle = 2.0 * pi / (numEntries - 1.0)
for p in range(0, numPts):
    y, x = (edgePoints[p])[0], (edgePoints[p])[1]

    # The angle is in the interval -pi,+pi
    ang = angleTemplate[y,x] + pi
    entryIndex = int(ang/deltaAngle)
    entry = rTable[entryIndex]
    entry.append((y-refPoint[0], x-refPoint[1]))
```

### Code 5.11 Constructing the R-Table

Code 5.12 shows the implementation of the gathering process of the GHT by using the Cartesian definition in Equation (5.76). We apply the Canny edge detection operator to the input image and the result is stored in the arrays `magnitude` and `angle`. The `magnitude` value is used to determine if a pixel is an edge and the `angle` is used to locate an entry in the R-Table. The `entryIndex` defines the index on the table and the `row` variable is a list that contains the table's entry. Each entry in the row increases a cell in the accumulator. The maximum number of votes occurs at the location of the original reference point. After all edge points have been inspected, the location of the shape is given by the maximum of the accumulator array.

Note that if we want to try other values for rotation and scale, then it is necessary to compute a table  $\gamma(\lambda, \rho)$  for all potential values. However, this can be avoided by considering that  $\gamma(\lambda, \rho)$  can be computed from  $\gamma(1,0)$ . That is, if we want to accumulate evidence for  $\gamma(\lambda, \rho)$ , then we use the entry indexed by  $\hat{\phi}'_i + \rho$  and we rotate and scale the vector  $\gamma(1,0)$ . That is,

$$\begin{aligned} \gamma_x(\lambda, \rho) &= \lambda(\gamma_x(1,0)\cos(\rho) - \gamma_y(1,0)\sin(\rho)) \\ \gamma_y(\lambda, \rho) &= \lambda(\gamma_x(1,0)\sin(\rho) + \gamma_y(1,0)\cos(\rho)) \end{aligned} \quad (5.87)$$

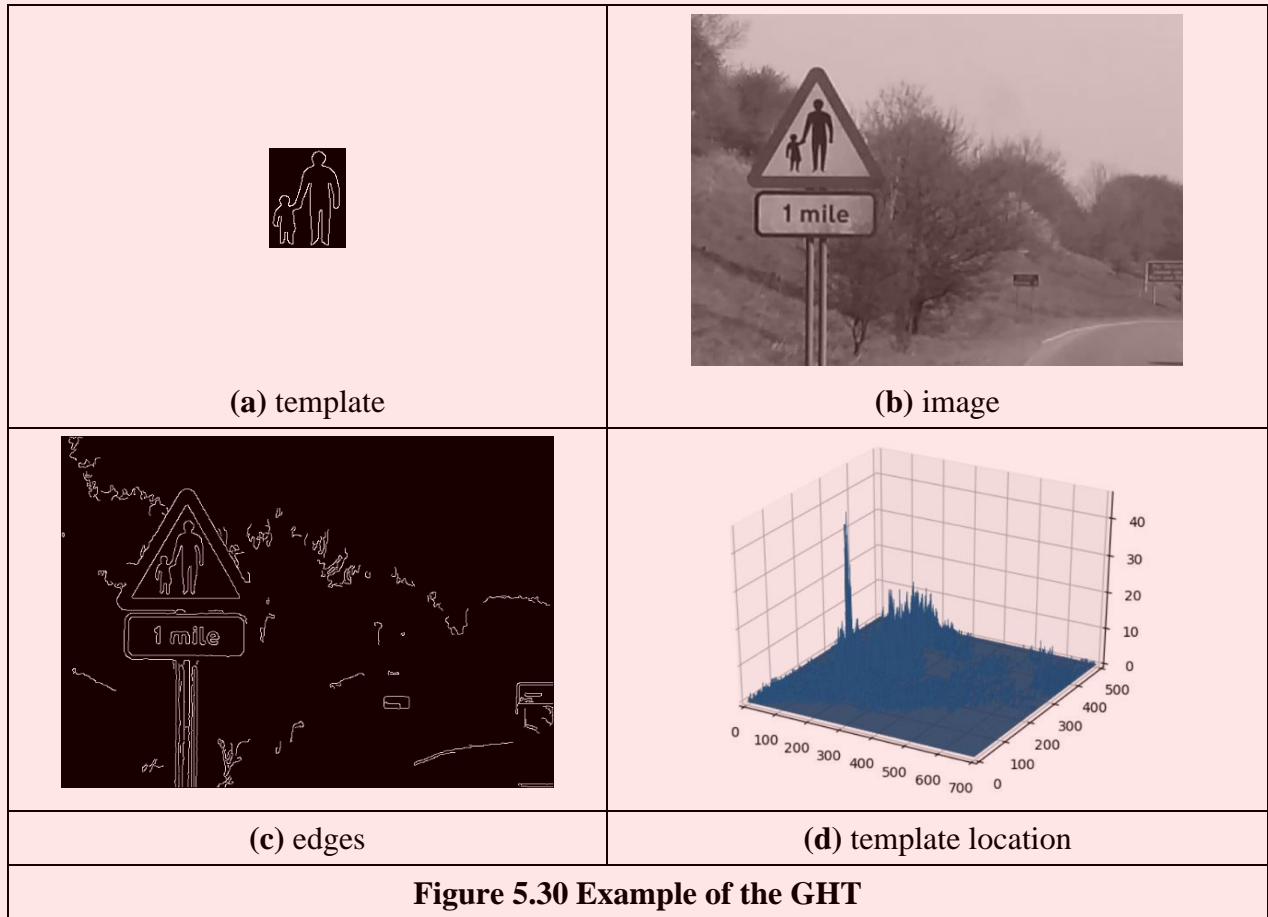
In the case of the polar form, the angle and magnitude need to be defined according to Equation (5.84).

```
# Gather evidence of the template in the image
accumulator = createImageF(width, height)
maxSegmentLength = 0
for x,y in itertools.product(range(0, width), range(0, height)):
    if magnitude[y,x] != 0:
        # The angle is in the interval -pi,+pi
        ang = angle[y,x] + pi
        entryIndex = int(ang/deltaAngle)
        row = rTable[entryIndex]
        numPts = len(row)
        for p in range(0, numPts):
            x0, y0 = x - (row[p])[1], y - (row[p])[0]
            if y0>0 and x0>0 and y0<height and x0<width:
                accumulator[y0][x0] += 1
```

### Code 5.12 Implementing the GHT

The application of the GHT to detect an arbitrary shape with unknown translation is illustrated in Figure 5.30. We constructed an R-table from the template shown in Figure 5.30(a). The table contains 180 rows. The accumulator in Figure 5.30(d) was obtained by applying the GHT to the image in Figure 5.30(b). Since the table was obtained from a shape with the same scale and rotation than the primitive in the image, then the GHT produces an accumulator with a clear peak at the centre of mass of the shape. The edges used to gather evidence are shown in Figure 5.30(c).

Although the example in Figure 5.30 shows that the GHT is an effective method for shape extraction, there are several inherent difficulties in its formulation [Grimson90][Aguado00b]. The most evident problem is that the table does not provide an accurate representation when objects are scaled and translated. This is because the table implicitly assumes that the curve is represented in discrete form. Thus, the GHT maps a discrete form into a discrete parameter space. Additionally, the transformation of scale and rotation can induce other discretisation errors. This is because when discrete images are mapped to be larger, or when they are rotated, loci which are unbroken sets of points rarely map to unbroken sets in the new image. Another important problem is the excessive computations required by the four dimensional parameter space. This makes the technique impractical. Also, the GHT is clearly dependent on the accuracy of directional information. By these factors, the results provided by the GHT can become less reliable. A solution is to use of an analytic form instead of a table [Aguado98]. This avoids discretisation errors and makes the technique more reliable. This also allows the extension to affine or other transformations. However, this technique requires solving for the point  $v(\theta)$  in an analytic way increasing the computational load. A solution is to reduce the number of points by considering characteristics points defined as points of high curvature. However this still requires the use of a four dimensional accumulator. An alternative to reduce this computational load is to include the concept of invariance in the GHT mapping.



#### 5.5.6.4 Invariant GHT

The problem with the GHT (and other extensions of the HT) is that they are very general. That is, the HT gathers evidence for a single point in the image. However, a point on its own provides little information. Thus, it is necessary to consider a large parameter space to cover all the potential shapes defined by a given image point. The GHT improves evidence gathering by considering a point and its gradient direction. However, since gradient direction changes with rotation, then the evidence gathering is improved in terms of noise handling, but little is done about computational complexity.

In order to reduce computational complexity of the GHT, we can consider replacing the gradient direction by another feature. That is, by a feature that is not affected by rotation. Let us explain this idea in more detail. The main aim of the constraint in Equation (5.79), is to include gradient direction to reduce the number of votes in the accumulator by identifying a point  $v(\theta)$ . Once this point is known, then we obtain the displacement vector  $\gamma(\lambda, \rho)$ . However, for each value of rotation, we have a different point in  $v(\theta)$ . Now let us replace that constraint in Equation (5.78) by a constraint of the form

$$Q(\omega_i) = Q(v(\theta)) \tag{5.88}$$

The function  $Q$  is said to be invariant and it computes a feature at the point. This feature can be, for example, the colour of the point, or any other property that does not change in the model and in the image. By considering Equation (5.88), Equation (5.79) is redefined as

$$Q(\omega_i) - Q(v(\theta)) = 0 \tag{5.89}$$

That is, instead of searching for a point with the same gradient direction, we will search for the point with the same invariant feature. The advantage is that this feature will not change with rotation or scale, so we only require a 2D space to locate the shape. The definition of  $Q$  depends on the application and the type of transformation. The most general invariant properties can be obtained by considering geometric definitions. In the case of rotation and scale changes (i.e., similarity transformations) the fundamental invariant property is given by the concept of angle.

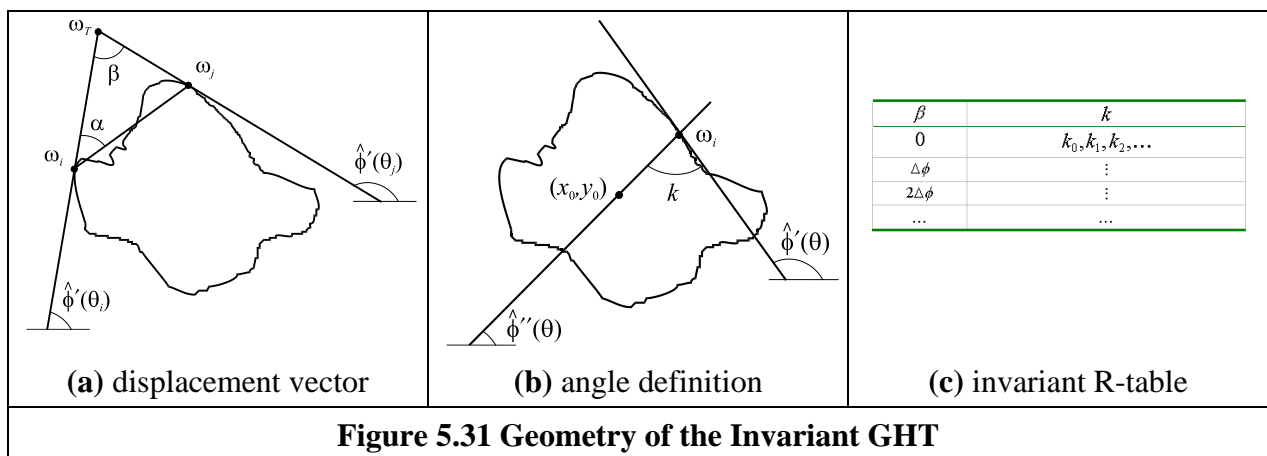
An angle is defined by three points and its value remains unchanged when it is rotated and scaled. Thus, if we associate to each edge point  $\omega_i$  a set of other two points  $\{\omega_j, \omega_T\}$ , we can compute a geometric feature that is invariant to similarity transformations. That is,

$$Q(\omega_i) = \frac{\omega_{xj}\omega_{yi} - \omega_{xi}\omega_{yj}}{\omega_{xi}\omega_{xj} + \omega_{yi}\omega_{yj}} \tag{5.90}$$

where  $\omega_{xn}$  and  $\omega_{yn}$  are the  $x$  and the  $y$  coordinates of point  $n$ . Equation (5.90) defines the tangent of the angle at the point  $\omega_T$ . In general, we can define the points  $\{\omega_j, \omega_T\}$  in different ways. An alternative geometric arrangement is shown in Figure 5.31(a). Given the points  $\omega_i$  and a fixed angle  $\vartheta$ , then we determine the point  $\omega_j$  such that the angle between the tangent line at  $\omega_i$  and the line that joins the points is  $\vartheta$ . The third point is defined by the intersection of the tangent lines at  $\omega_i$  and  $\omega_j$ . The tangent of the angle  $\beta$  is defined by Equation (5.90). This can be expressed in terms of the points and its gradient directions as

$$Q(\omega_i) = \frac{\phi'_i - \phi'_j}{1 + \phi'_i\phi'_j} \tag{5.91}$$

We can replace the gradient angle in the R-table, by the angle  $\beta$  whose tangent is defined by Equation (5.91). The form of the new invariant table is shown in Figure 5.31(c). Since the angle  $\beta$  does not change with rotation or change of scale, then we do not need to change the index for each potential rotation and scale. However, the displacement vectors changes according to rotation and scale (i.e., Equation (5.87)) that. Thus, if we want an invariant formulation, we must also change the definition of the position vector.



In order to locate the reference point  $b$  from an entry on the table, we can generalise the ideas presented in Figure 27(a) and Figure 5.29(a). Figure 5.31(b) shows this generalization. As in the case of the circle and ellipse, we can locate the centre of a shape by considering a line of votes that



passes through the centre point  $b$ . This line was determined for the circle and ellipse by the angle  $\phi_i''$ . In order to use it in general shapes we will need do two things. First, we will find an invariant definition of this value. Secondly, we will need to include it on the GHT table.

We can develop Equation (5.75) as

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \omega_{xi} \\ \omega_{yi} \end{bmatrix} + \lambda \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} x(\theta) \\ y(\theta) \end{bmatrix} \quad (5.92)$$

Thus, Equation (5.62) generalises to

$$\phi_i'' = \frac{\omega_{yi} - y_0}{\omega_{xi} - x_0} = \frac{[-\sin(\rho) \quad \cos(\rho)]y(\theta)}{[\cos(\rho) \quad \sin(\rho)]x(\theta)} \quad (5.93)$$

By some algebraic manipulation, we have that

$$\phi_i'' = \tan(\xi - \rho) \quad (5.94)$$

where

$$\xi = \frac{y(\theta)}{x(\theta)} \quad (5.95)$$

In order to define  $\phi_i'$  we can consider the tangent angle at the point  $\omega_i$ . By considering the derivative of Equation (5.74) we have that

$$\phi_i' = \frac{[-\sin(\rho) \quad \cos(\rho)]y'(\theta)}{[\cos(\rho) \quad \sin(\rho)]x'(\theta)} \quad (5.96)$$

Thus,

$$\phi_i' = \tan(\phi - \rho) \quad (5.97)$$

where

$$\phi = \frac{y'(\theta)}{x'(\theta)} \quad (5.98)$$

By considering Equation (5.94) and Equation (5.96) we define

$$\hat{\phi}_i'' = k + \hat{\phi}_i' \quad (5.99)$$

The important point in this definition is that the value of  $k$  is invariant to rotation. Thus, if we use this value in combination with the tangent at a point we can have an invariant characterisation. In order to see that  $k$  is invariant, we solve it for Equation (5.99). That is,

$$k = \hat{\phi}_i' - \hat{\phi}_i'' \quad (5.100)$$

Thus,

$$k = \xi - \rho - (\phi - \rho) \quad (5.101)$$

That is,

$$k = \xi - \phi \quad (5.102)$$

That is independent of rotation. The definition of  $k$  has a simple geometric interpretation illustrated in Figure 5.25 Geometry of the Angle of the First and Second Directional Derivatives(b).

In order to obtain an invariant GHT, it is necessary to know for each point  $\omega_i$ , the corresponding point  $\nu(\theta)$  and then compute the value of  $\phi_i''$ . Then evidence can be gathered by the line in Equation (5.93). That is,

$$y_0 = \phi_i''(x_0 - \omega_{xi}) + \omega_{yi} \quad (5.103)$$

In order to compute  $\phi_i''$  we can obtain  $k$  and then use Equation (5.101). In the standard tabular form the value of  $k$  can be pre-computed and stored as function of the angle  $\beta$ .

```
# Find the pairs of points in the template according to alpha angle
pairPoints = []
numPts = len(edgePoints)
for p in range(0, numPts):
    y1, x1 = (edgePoints[p])[0], (edgePoints[p])[1]
    # We are looking for two points along the line with slope m
    m = tan(angleTemplate[y1,x1] - alpha)
    if m>-1 and m<1:
        if x1 < refPoint[1]: xi, xf, step = x1 + minimaDistPoints,widthTemplate, 1
        else: xi, xf, step = x1 - minimaDistPoints, 0, -1
        for x2 in range(xi, xf, step):
            y2 = int(y1 - m * (x2 - x1))
            if y2 > 0 and y2 < heightTemplate and magnitudeTemplate[y2,x2] != 0:
                pairPoints.append((y2,x2,y1,x1))
                break
    else:
        m = 1.0/m
        if y1 < refPoint[0]: yi, yf, step = y1 + minimaDistPoints,heightTemplate, 1
        else: yi, yf, step = y1 - minimaDistPoints, 0, -1
        for y2 in range(yi, yf, step):
            x2 = int(x1 - m * (y2 - y1))
            if x2 > 0 and x2 < widthTemplate and magnitudeTemplate[y2,x2] != 0:
                pairPoints.append((y2,x2,y1,x1))
                break

# Build table (k,c) from each pair of points
rTable = [[] for entryIndex in range(numEntries)]
deltaAngle = pi / (numEntries - 1.0)
numPairs = len(pairPoints)
for pair in range(0, numPairs):
    y2, x2 = (pairPoints[pair])[0], (pairPoints[pair])[1]
    y1, x1 = (pairPoints[pair])[2], (pairPoints[pair])[3]
    # Compute beta
    phi1, phi2 = tan(-angleTemplate[y1,x1]), tan(-angleTemplate[y2,x2])

    if 1.0+phi1*phi2 != 0: beta = atan((phi1-phi2)/(1.0+phi1*phi2))
    else: beta=1.57

    # Compute k
    if x1- refPoint[1] !=0: m = atan(-float(y1-refPoint[0])/(x1-refPoint[1]))
    else: m =1.57

    k = angleTemplate[y1,x1] - m
    # Scale
    distCentre = sqrt((y1-refPoint[0])*(y1-refPoint[0]) + \
                      (x1-refPoint[1])*(x1-refPoint[1]))
    distPoints = sqrt((y2-y1)*(y2-y1) + (x2-x1)*(x2-x1))
    c = distCentre/distPoints
    # Insert in the table. The angle is in the interval -pi/2 to pi/2
    entryIndex = int((beta+(pi/2.0))/deltaAngle)
    entry = rTable[entryIndex]
    entry.append((k,c))
```

**Code 5.13 Constructing the Invariant R-Table**

Code 5.13 illustrates the implementation for obtaining the invariant R-Table. This code searches for two points along the line defined by the value of  $\alpha$ . In the code `alpha` is set to  $\pi/2$ . Each element of the table stores a single value computed according to Equation (5.100). The more cumbersome part of the code is to search for the point  $\omega_j$ . We search in two directions from  $\omega_i$  and we stop once an edge point has been located. This search is performed by tracing a line. The trace is dependent on the slope. When the slope is between -1 and +1 we then determine a value of  $y$  for each value of  $x$ , otherwise we determine a value of  $x$  for each value of  $y$ . The table is constructed by considering each pair of points. The value `beta` is obtained according to Equation (5.89). The value  $k$  by following Equation (5.102). The scale  $c$  is simply the ratio between the distance of the first point to the reference point and the distance between the points. The value `beta` is then used to find the index in the table where to insert the pair  $(k, c)$ .

Code 5.14 illustrates the evidence gathering process according to Equation (5.103). The first part of the process is to find for each edge point  $(x_1, y_1)$  all the points along the line defined by Equation (5.94). This process is similar to the first part of Code 5.13 and the results are stored in the list `secondPoints`. Each point  $(x_1, y_1)$  is paired with each point in the `secondPoints` list to define the pairs  $(x_1, y_1), (x_2, y_2)$ . For each pair we can compute a value `beta` according to Equation (5.91). This defines an entry on the table. The data  $(k, c)$  is recovered from the table and it is used to compute the slope of the angle defined in Equation (5.99). The slope of the line is obtained from  $k$  and the distance between the point and the centre is determined by  $c$ . The accumulator is then incremented by a small segment of three pixels in the location of the reference point. We use a three pixels segment to account for errors in the computation of the gradient, the discretisation of the table and the discretisation of the accumulator. We should expect the location of the centre to be close but not exactly at the position obtained from the table values. The increase in the accumulator includes a weighting obtained by the distance to the expected reference point position.

Figure 5.32 shows an example of the accumulator obtained by the implementation of Code 5.13. Figure 5.32(a) shows the template used in this example. This template was used to construct the R-Table in Code 5.13. The R-Table was used to accumulate evidence when searching for the template in the image in Figure 5.32(b). Figure 5.32(d) shows the result of the evidence gathering process. We can observe a peak in the location of the object. However, this accumulator contains some noise. The noise is produced since rotation and scale change the value of the computed gradient. Thus, the line of votes is only approximated. Another problem is that pairs of points  $\omega_i$  and  $\omega_j$  might not be found in an image, thus the technique is more sensitive to occlusion and noise than the GHT.

```

# Gather evidence
numPts = len(secondPoints) > 0
for ptIndex in range(0, numPts):
    secondPoint = secondPoints[ptIndex]
    y2, x2 = secondPoint[0], secondPoint[1]
    distPoints = sqrt((y2-y1)*(y2-y1) + (x2-x1)*(x2-x1))

    # Compute beta
    phi1, phi2 = tan(-angle[y1,x1]), tan(-angle[y2,x2])
    if 1.0+phi1*phi2 != 0:
        beta = atan((phi1-phi2)/(1.0+phi1*phi2))
    else:
        beta=1.57
    # Find entry in table
    entryIndex = int((beta+(pi/2.0))/deltaAngle)

    row = rTable[entryIndex]
    numEntriesinRow = len(row)

    for kIndex in range(0, numEntriesinRow):
        k, c = (row[kIndex])[0], (row[kIndex])[1]
        distCentre = c * distPoints
        m = tan(angle[y1,x1] - k)
        if m>-1 and m<1:
            for x in range(0, width):
                y = int(y1 - m * (x - x1))
                d = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1))
                if y > 0 and y < height and abs(d-distCentre) < 3:
                    accumulator[y,x] += 3.0 - abs(d-distCentre)
        else:
            m = 1.0/m
            for y in range(0, height):
                x = int(x1 - m * (y - y1))
                d = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1))
                if x > 0 and x < width and abs(d-distCentre) < 3:
                    accumulator[y,x] += 3.0 - abs(d-distCentre)

```

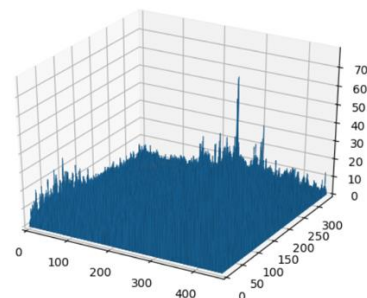
**Code 5.14 Implementing the Invariant GHT**



(a) template



(b) image



(c) edges	(d) template location
<b>Figure 5.32 Applying the Invariant GHT</b>	

### 5.5.7 Other Extensions to the HT

The motivation for extending the HT is clear: keep the performance, but improve the speed. There are other approaches to reduce the computational load of the HT. These approaches aim to improve speed and reduce memory focusing on smaller regions of the accumulator space. These approaches have included: the *Fast HT* [Li86] which successively splits the accumulator space into quadrants and continues to study the quadrant with most evidence; the *Adaptive HT* [Illingworth87] which uses a fixed accumulator size to iteratively focus onto potential maxima in the accumulator space; the *Randomised HT* [Xu90] and the *Probabilistic HT* [Kälviäinen95] which use a random search of the accumulator space; and other pyramidal techniques. One main problem with techniques which do not search the full accumulator space, but a reduced version to save speed, is that the wrong shape can be extracted [Princen89], a problem known as *phantom shape location*. These approaches can also be used (with some variation) to improve speed of performance in template matching. There have been many approaches aimed to improve performance of the HT and of the GHT.

There has been a comparative study on the GHT (including efficiency) [Kassim99] and alternative approaches to the GHT include two *Fuzzy HTs*: [Philip91] which [Sonka94] includes uncertainty of the perimeter points within a GHT structure and [Han94] which approximately fits a shape but which requires application-specific specification of a fuzzy membership function. There have been two major reviews of the state of research in the HT [Illingworth88] [Leavers93] (but they are rather dated now) and a textbook [Leavers92] which cover many of these topics. The analytic approaches to improving the HTs' performance use mathematical analysis to reduce size, and more importantly dimensionality, of the accumulator space. This concurrently improves speed. A review of HT based techniques for circle extraction [Yuen90] covered some of the most popular techniques available at the time.

## 5.6 Further Reading

It is worth noting that research has focussed on shape extraction by combination of low-level features, Section 5.4, rather than on HT based approaches. The advantages of the low-level feature approach are simplicity, in that the features exposed are generally less complex than the variants of the HT. There is also a putative advantage in speed, in that simpler approaches are invariably faster than those that are more complex. Any advantage in respect of performance in noise and occlusion is yet to be established. The HT approaches do not (or do not yet) include machine learning approaches whose potency is perhaps achieved by the techniques which use low level features. The use of machine learning also implies a need for training, but there is a need to generate some form of template for the HT or template approaches. An over-arching premise of this text is that there is no panacea and as such there is a selection of techniques as there are for feature extraction, and some of the major approaches have been covered in this Chapter.

In terms of performance evaluation, it is worth noting the PASCAL Visual Object Classes (VOC) Challenge [Everingham10] which is a benchmark in visual object category recognition and detection. The PASCAL consortium <http://host.robots.ox.ac.uk/pascal/VOC/> aims to provide standardised databases for object recognition; to provide a common set of tools for accessing and managing the database annotations; and to conduct challenges, which evaluate performance on

object class recognition. This provides evaluation data and mechanisms and thus describing many advances in recognising objects from a number of visual object classes in realistic scenes.

The majority of further reading in finding shapes concerns papers, many of which have already been referenced, especially in the newer techniques. An excellent survey of the techniques used for feature extraction (including template matching, deformable templates etc.) can be found in [Trier96]. A recent survey of the HT techniques can be found in [Mukhopadhyay15]. Few of the textbooks devote much space to shape extraction except *Shape Classification and Analysis* [Costa09] and *Template Matching Techniques in Computer Vision* [Brunelli09], sometimes dismissing it in a couple of pages. This rather contrasts with the volume of research there has been in this area, and the Hough transform finds increasing application as computational power continues to increase (and storage cost reduces). Other techniques use a similar evidence gathering process to the HT. These techniques are referred to as Geometric Hashing and Clustering Techniques [Lamdan88] [Stockman87]. In contrast with the HT, these techniques do not define an analytic mapping, but they gather evidence by grouping a set of features computed from the image and from the model.

There are deep learning replacing interest point detectors [Zheng18] and deep learning based object detectors [Ren15, Redmon16]. There are also newer approaches that perform segmentation based on evidence gathering, e.g. Hough-CNN [Milletari17], which perhaps can handle better the noise in medical images. Time will tell on these new approaches, and some are described in Chapter 12. Essentially, this Chapter has focussed on shapes, which can in some form have a fixed appearance whether it is exposed by a template, a set of keypoints or by a description of local properties. In order to extend the approaches to shapes with a less constrained description, and rather than describe such shapes by constructing a library of their possible appearances, we require techniques for deformable shape analysis, as we shall find in the next Chapter.

## 5.7 Chapter 5 References

- [Aguado96a] Aguado, A. S., *Primitive Extraction via Gathering Evidence of Global Parameterised Models*, PhD Thesis, University of Southampton, 1996
- [Aguado96b] Aguado, A. S., Montiel, E., and Nixon, M. S., On Using Directional Information for Parameter Space Decomposition in Ellipse Detection, *Pattern Recognition* **28**(3), pp 369-381, 1996
- [Aguado98] Aguado, A. S., Nixon, M. S., and Montiel, M. E., Parameterising Arbitrary Shapes via Fourier Descriptors for Evidence-Gathering Extraction, *Computer Vision and Image Understanding*, **69**(2), pp 202-221, 1998
- [Aguado00a] Aguado, A. S., Montiel, E., and Nixon, M. S., On the Intimate Relationship Between the Principle of Duality and the Hough Transform, *Proceedings of the Royal Society A*, **456**, pp 503-526, 2000
- [Aguado00b] Aguado, A. S., Montiel, E., and Nixon, M. S., Bias Error Analysis of the Generalised Hough Transform, *Journal of Mathematical Imaging and Vision*, **12**, pp 25-42, 2000
- [Altman84] Altman, J., and Reitbock, H. J. P., A Fast Correlation Method for Scale- and Translation-Invariant Pattern Recognition, *IEEE Trans. on PAMI*, **6**(1), pp 46-57, 1984
- [Arandjelović12] Arandjelović, R. and Zisserman, A., Three things everyone should know to improve object retrieval. Proc. *IEEE CVPR*, pp 2911-2918, 2012
- [Arbab-Zavar11] Arbab-Zavar, B., and Nixon, M. S., On Guided Model-Based Analysis for Ear Biometrics, *Computer Vision and Image Understanding*, **115**, pp 487–502, 2011

- [Ballard81] Ballard, D. H., Generalising the Hough Transform to Find Arbitrary Shapes, *CVGIP*, **13**, pp111-122, 1981
- [Bay08] Bay, H., Eas, A., Tuytelaars, T., and Van Gool, L., Speeded-Up Robust Features (SURF), *Computer Vision and Image Understanding*, **110**(3), pp 346-359, 2008
- [Bracewell86] Bracewell, R. N., *The Fourier Transform and its Applications*, 2<sup>nd</sup> Edition, McGraw-Hill Book Co, Singapore, 1986
- [Bresenham65] Bresenham, J. E., Algorithm for Computer Control of a Digital Plotter, *IBM Systems Journal*, **4**(1), pp 25-30, 1965
- [Bresenham77] Bresenham, J. E., A Linear Algorithm for Incremental Digital Display of Circular Arcs, *Comms. of the ACM*, **20**(2), pp 750-752, 1977
- [Brown83] Brown, C. M., Inherent bias and noise in the Hough transform, *IEEE Trans. on PAMI*, **5**, pp 493-505, 1983
- [Brunelli09] Brunelli, R., *Template Matching Techniques in Computer Vision*, Wiley, Chichester UK, 2009
- [Bustard10] Bustard, J. D., and Nixon, M. S., Toward Unconstrained Ear Recognition From Two-Dimensional Images, *IEEE Trans SMC(A)*, **40**(3), pp 486-494, 2010
- [Calonder10] Calonder, M., Lepetit, V., Strecha, C. and Fua, P., BRIEF: Binary Robust Independent Elementary Features. Proc. *ECCV*, pp. 778-792, 2010
- [Casasent77] Casasent, D., and Psaltis, D., New Optical Transforms for Pattern Recognition, *Proceedings of the IEEE*, **65**(1), pp 77-83, 1977
- [Costa09] Costa, L. F., and Cesar, L. M., *Shape Classification and Analysis*, CRC Press (Taylor & Francis), Boca Raton FL USA, 2<sup>nd</sup> Edition, 2009
- [Dalal05] Dalal, N., and Triggs, B., Histograms of Oriented Gradients for Human Detection, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, **2**, pp 886-893, 2005
- [Datta 08] Datta, R., Joshi, D., Li, J., and Wang, J. Z. 2008. Image retrieval: Ideas, Influences, and Trends of the New Age. *ACM Comput. Surv.* **40**(2), Article 5. April 2008
- [Deans81] Deans, S. R., Hough transform from the Radon transform, *IEEE Trans. on PAMI*, **13**, pp 185-188, 1981
- [Duda72] Duda, R. O. and Hart, P. E., Use of the Hough Transform to Detect Lines and Curves in Pictures, *Comms. of the ACM*, **15**, pp 11-15, 1972
- [Everingham10] Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A., The PASCAL Visual Object Classes (VOC) Challenge, *International Journal of Computer Vision*, **88**(2), pp 303-338, 2010
- [Gerig86] Gerig, G., and Klein, F., Fast Contour Identification through Efficient Hough Transform and Simplified Interpretation Strategy, *Proc. 8<sup>th</sup> Int. Conf. Pattern Recog*, pp 498-500, 1986
- [Grimson90] Grimson, W. E. L., and Huttenglocher, D. P., On the Sensitivity of the Hough Transform for Object Recognition, *IEEE Trans. on PAMI*, **12**, pp 255-275, 1990
- [Han94] Han, J. H., Koczy, L. T. and Poston, T., Fuzzy Hough Transform, *Pattern Recog. Lett.*, **15**, pp 649-659, 1994
- [Hecker94] Hecker, Y. C., and Bolle, R. M., On Geometric Hashing and the Generalized Hough Transform, *IEEE Trans. On Systems, Man and Cybernetics*, **24**, pp 1328-1338, 1994
- [Hough62] Hough, P. V. C., Method and Means for Recognising Complex Patterns, *US Patent 3069654*, 1962
- [Hurley08] Hurley D. J., Arbab-Zavar, B., and Nixon, M. S., The Ear as a Biometric, In A. Jain, P. Flynn and A. Ross Eds.: *Handbook of Biometrics*, pp 131-150, 2008

- [Illingworth87] Illingworth, J., and Kittler, J., The Adaptive Hough Transform, *IEEE Trans. on PAMI*, **9**(5), pp 690-697, 1987
- [Illingworth88] Illingworth, J., and Kittler, J., A Survey of the Hough Transform, *CVGIP*, **48**, pp 87-116, 1988
- [Kälviäinen95] Kälviäinen, H., Hirvonen, P., Xu, L., and Oja, E., Probabilistic and Non-probabilistic Hough Transforms: Overview and Comparisons, *Image and Vision Computing*, **13**(4), pp 239-252, 1995
- [Kassim99] Kassim, A. A., Tan, T., Tan K. H., A Comparative Study of Efficient Generalised Hough Transform Techniques, *Image and Vision Computing*, **17**(10), pp. 737-748, 1999
- [Kimme75] Kimme, C., Ballard, D., and Sklansky, J., Finding Circles by an Array of Accumulators, *Comms. ACM*, **18**(2), pp 120-122, 1975
- [Kiryati91] Kiryati, N., and Bruckstein, A. M., Antialiasing the Hough transform, *CVGIP: Graphical Models and Image Processing*, **53**, pp 213-222, 1991
- [Lamdan88] Lamdan, Y., Schawatz, J., and Wolfon, H., Object Recognition by Affine Invariant Matching, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, pp 335-344, 1988
- [Leavers92] Leavers, V., *Shape Detection in Computer Vision using the Hough Transform*, London: Springer-Verlag, 1992
- [Leavers93] Leavers, V., Which Hough Transform, *CVGIP: Image Understanding*, **58**, pp 250-264, 1993
- [Li86] Li, H., and Lavin, M. A., Fast Hough Transform: a Hierarchical Approach, *CVGIP*, **36**, pp 139-161, 1986
- [Lienhart03] Lienhart, R., Kuranov, A., and Pisarevsky, V., Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection, *LNCS* **2781**, pp. 297–304, 2003
- [Lowe04] Lowe, D. G., Distinctive Image Features from Scale-Invariant Key points, *International Journal of Computer Vision*, **60**(2), pp 91-110, 2004
- [Merlin75] Merlin, P. M. and Farber, D. J., A Parallel Mechanism for Detecting Curves in Pictures, *IEEE Trans. on Computers*, **24**, pp 96-98, 1975
- [Mikolajczyk05] Mikolajczyk, K., and Schmid, C., A Performance Evaluation of Local Descriptors, *IEEE Trans. on PAMI*, **27**(10), pp 1615- 1630, 2005
- [Milletari117] Milletari, F., Ahmadi, S.A., Kroll, C., Plate, A., Rozanski, V., Maiostre, J., Levin, J., Dietrich, O., Ertl-Wagner, B., Bötzel, K. and Navab, N., Hough-CNN: deep learning for segmentation of deep brain regions in MRI and ultrasound. *Computer Vision and Image Understanding*, 164, pp 92-102, 2017
- [Mukherjee15] Mukherjee, D., Wu, Q. J. and Wang, G., A comparative experimental study of image feature detectors and descriptors. *Machine Vis. and Apps.*, **26**(4), pp.443-466, 2015
- [Mukhopadhyay15] Mukhopadhyay, P. and Chaudhuri, B. B., A survey of Hough Transform. *Pattern Recognition*, **48**(3), pp.993-1010, 2015
- [O'Gorman76] O'Gorman, F., and Clowes, M. B., Finding Picture Edges Through Collinearity of Feature Points, *IEEE Trans. on Computers*, **25**(4), pp 449-456, 1976
- [Philip91] Philip, K. P., *Automatic Detection of Myocardial Contours in Cine Computed Tomographic Images*, PhD Thesis, Univ. Iowa USA, 1991
- [Princen89] Princen, J., Yuen, H. K., Illingworth, J., and Kittler, J., Properties of the Adaptive Hough Transform, *Proc. 6<sup>th</sup> Scandinavian Conf. on Image Analysis*, Oulu Finland, June, 1992
- [Princen92] Princen, J., Illingworth, J., and Kittler, J., A Formal Definition of the Hough Transform: Properties and Relationships, *J. Mathematical Imaging and Vision*, **1**, pp153-168, 1992



- [Redmon16] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A., You only look once: Unified, real-time object detection. Proc. IEEE CVPR, pp 779-788, 2016
- [Ren15] Ren, S., He, K., Girshick, R. and Sun, J., Faster r-cnn: Towards real-time object detection with region proposal networks. Proc. Advances in Neural Information Processing Systems pp 91-99, 2015
- [Rosenfeld69] Rosenfeld, A., *Picture Processing by Computer*, Academic Press, London UK, 1969
- [Ruble11] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G., ORB: An efficient alternative to SIFT or SURF, Proc. ICCV, pp 2564-2571, 2011
- [Schneiderman04] Schneiderman, H., and Kanade, T., Object detection using the statistics of parts, *International Journal of Computer Vision*, **56**(3), pp 151–177, 2004
- [Sivic03] Sivic, J., Zisserman, A., Video Google: A Text Retrieval Approach to Object Matching in Videos, *Proc. IEEE ICCV'03*, **2**, p.1470-1477, 2003
- [Sklansky78] Sklansky, J., On the Hough Technique for Curve Detection, *IEEE Trans. on Computers*, **27**, pp 923-926, 1978
- [Smeulders00] Smeulders, A. W. M., Worring, M., Santini, S., Gupta, A., and Jain, R., Content-Based Image Retrieval at the End of the Early Years, *IEEE Trans. on PAMI*, **22**(12), pp 1349-1378, 2000
- [Sonka94] Sonka, M., Hlavac, V., and Boyle, R., *Image Processing, Analysis and Computer Vision*, Chapman Hall, London UK, 1994
- [Stockman77] Stockman, G. C., and Agrawala, A. K., Equivalence of Hough Curve Detection to Template Matching, *Comms. of the ACM*, **20**, pp 820-822, 1977
- [Stockman87] Stockman, G., Object Recognition and Localization via Pose Clustering, *CVGIP*, **40**, pp361-387, 1987
- [Trier96] Trier, O. D., Jain, A. K., and Taxt, T., Feature Extraction Methods for Character Recognition – A Survey, *Pattern Recognition*, **29**(4), pp 641-662, 1996
- [Tuytelaars07] Tuytelaars, T., and Mikolajczyk, K., Local Invariant Feature Detectors: A Survey, *Foundations and Trends in Computer Graphics and Vision*, **3**(3), pp 177–280, 2007
- [Viola01] Viola, P., and Jones, M., Rapid Object Detection using a Boosted Cascade of Simple Features, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, **1**, pp 511-519, 2001
- [Viola04] Viola, P., and Jones, M. J., Robust real-time face detection, *International Journal of Computer Vision*, **57**(2), 137–154, 2004
- [Yuen90] Yuen, H. K., Princen, J., Illingworth, J., and Kittler, J., Comparative Study of Hough Transform Methods for Circle Finding, *Image and Vision Computing*, **8**(1), pp 71-77, 1990
- [Xu90] Xu, L., Oja, E., and Kultanen, P., A New Curve Detection Method: Randomised Hough Transform, *Pattern Recog. Lett.*, **11**, pp 331-338, 1990
- [Zhu06] Zhu, Q., Avidan, S., Yeh, M-C., and Cheng, K-T., Fast Human Detection Using a Cascade of Histograms of Oriented Gradients, *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, **2**, pp 1491-1498, 2006
- [Zafeiriou15] Zafeiriou, S., Zhang, C. and Zhang, Z., A survey on face detection in the wild: past, present and future. *CVIU*, **138**, pp 1-24, 2015
- [Zokai05] Zokai, S., and Wolberg G., Image Registration using Log-Polar Mappings for Recovery of Large-Scale Similarity and Projective Transformations, *IEEE Trans. IP*, **14**, pp 1422-1434, Oct. 2005
- [Zheng18] Zheng, L., Yang, Y. and Tian, Q., SIFT meets CNN: A decade survey of instance retrieval. *IEEE Trans. on PAMI*, **40**(5), pp.1224-1244, 2018