

*COMP3204/COMP6223: Computer Vision*

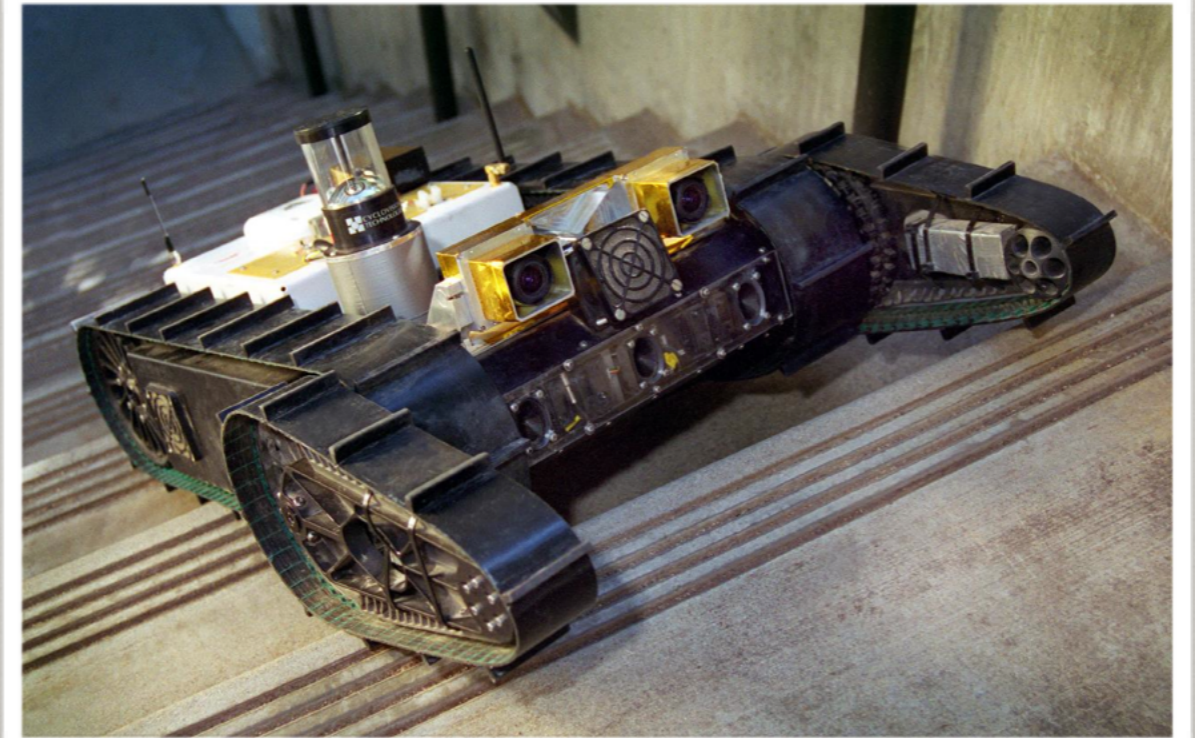
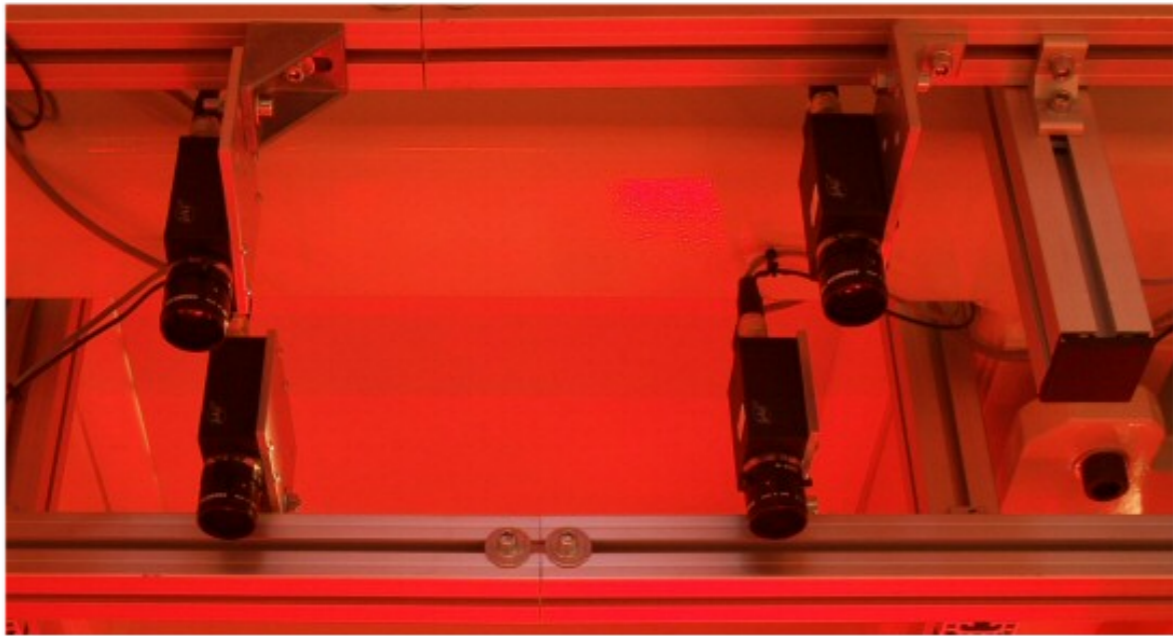
---

# Building machines that see

Jonathon Hare  
[jsh2@ecs.soton.ac.uk](mailto:jsh2@ecs.soton.ac.uk)

---

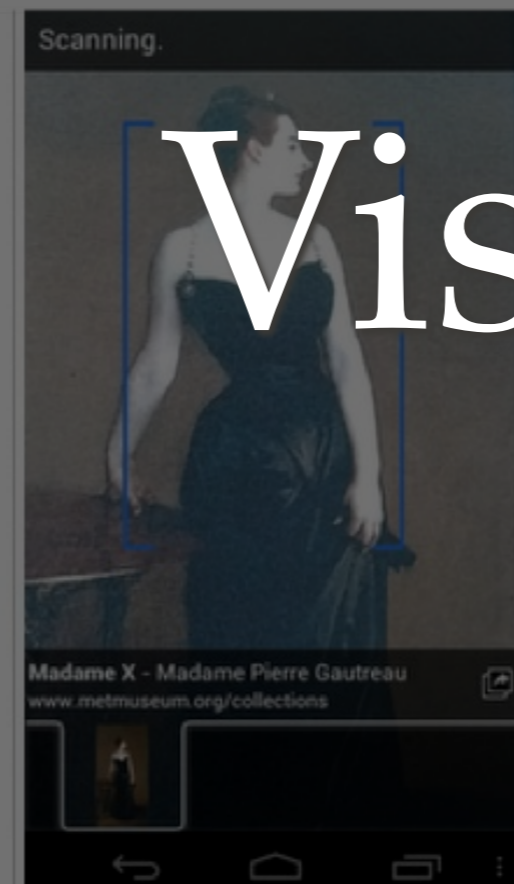
# Types of Computer *V*ision and their **Environment**

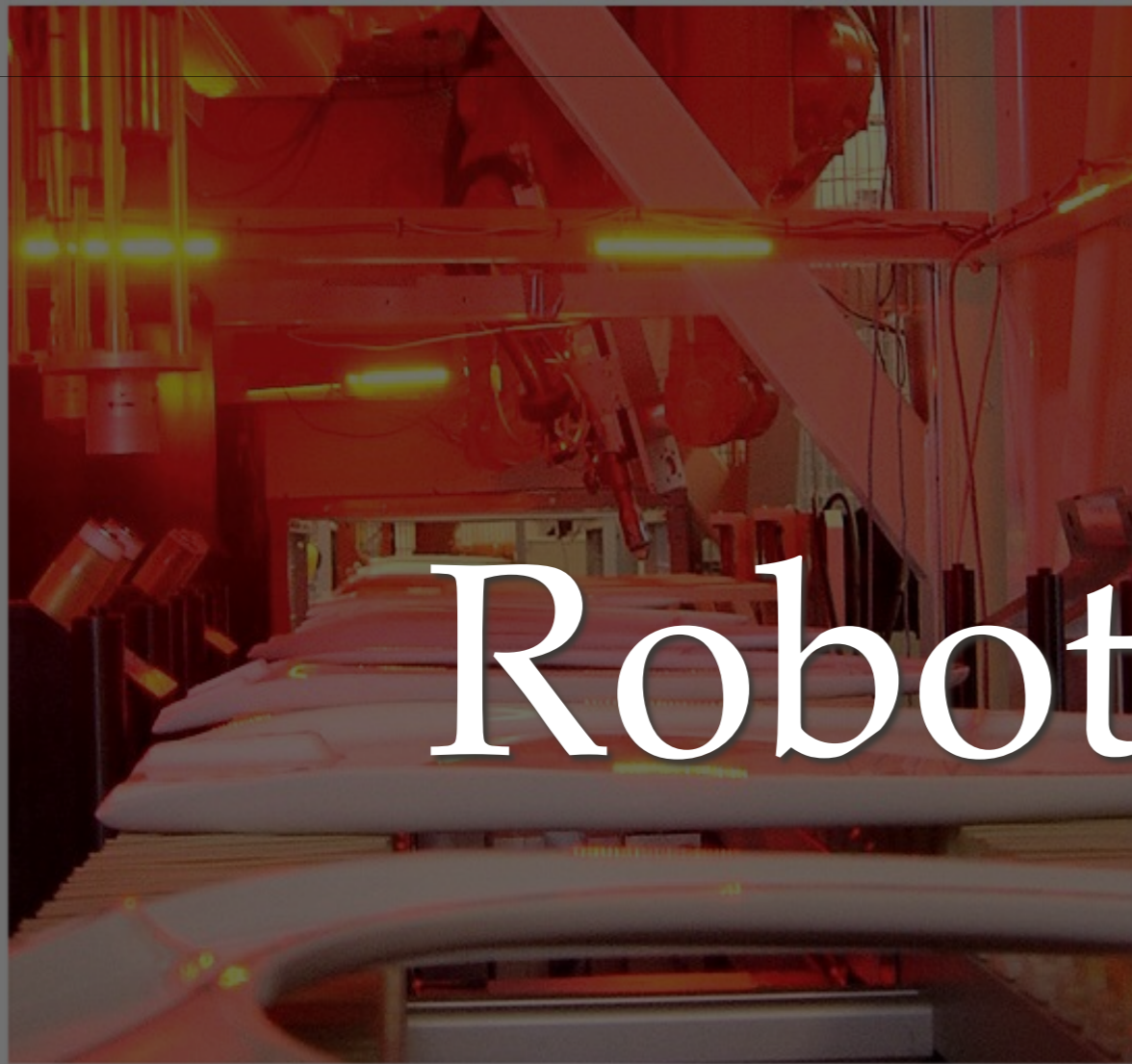
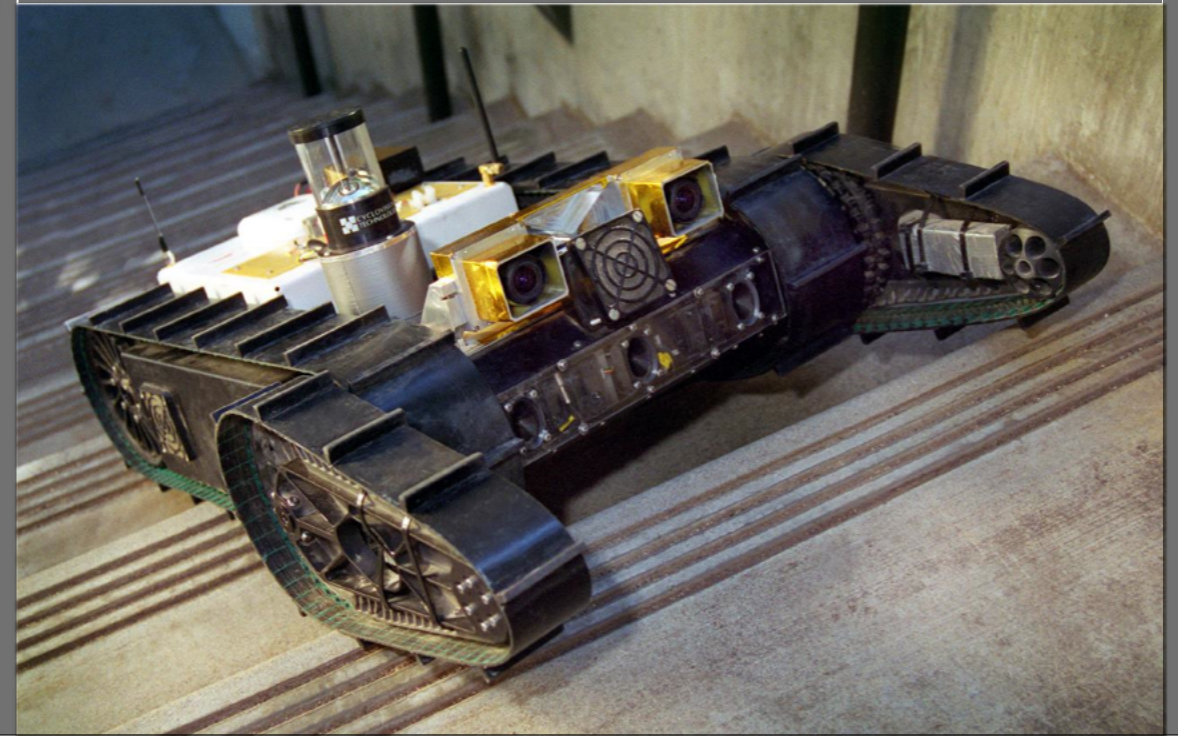




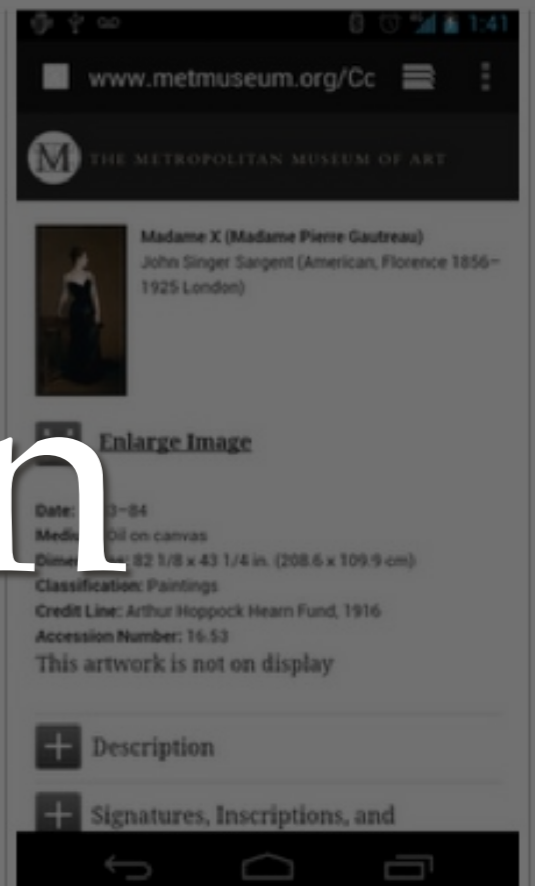
# Industrial

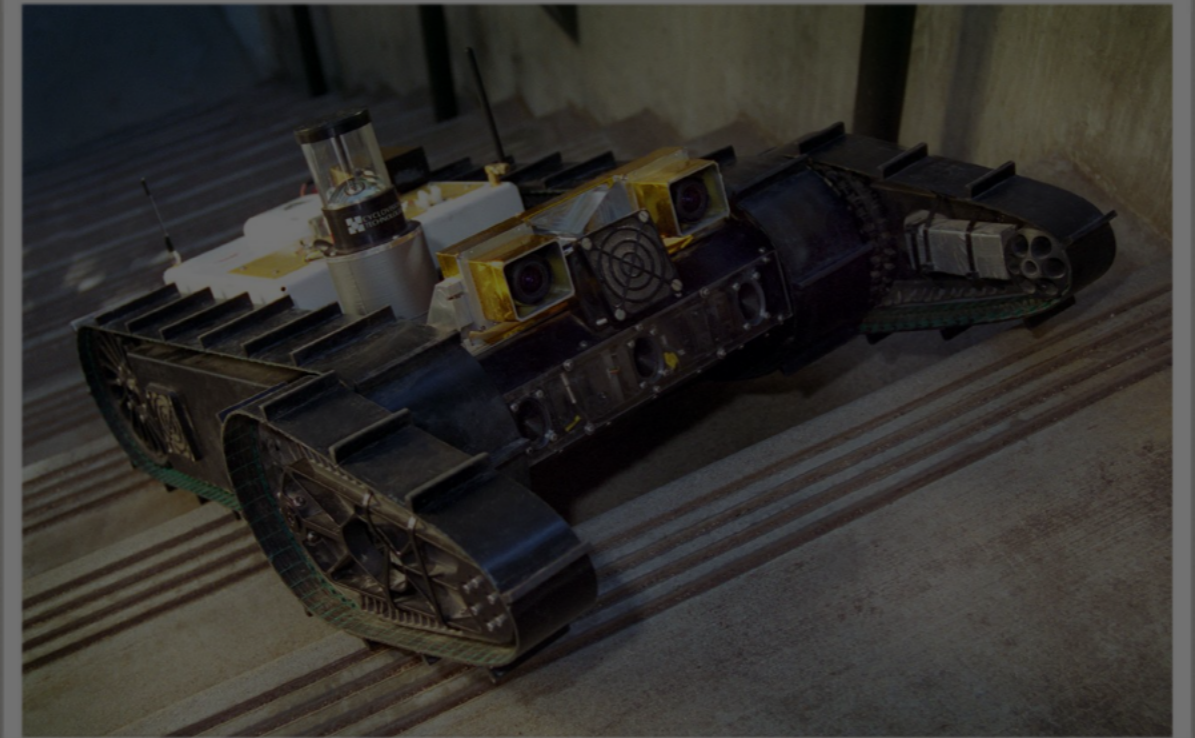
# Vision





# Robot Vision





Scanning.

Madame X - Madame Pierre Gautreau  
[www.metmuseum.org/collections](http://www.metmuseum.org/collections)

[www.metmuseum.org/Cc](http://www.metmuseum.org/Cc)

THE METROPOLITAN MUSEUM OF ART

**Madame X (Madame Pierre Gautreau)**  
John Singer Sargent (American, Florence 1856–1925 London)

[Enlarge Image](#)

Date: 1883–84  
Medium: Oil on canvas  
Dimensions: 82 1/8 x 43 1/4 in. (208.6 x 109.9 cm)  
Classification: Paintings  
Credit Line: Arthur Hoppock Hearn Fund, 1916  
Accession Number: 16.53  
This artwork is not on display

[+ Description](#)

[+ Signatures, Inscriptions, and](#)

What do all these systems have in  
common?

# Computer Vision Software

The screenshot displays the Eclipse IDE with a Java project named 'image-feature-extraction'. The main editor shows the 'EigenImages.java' file, which implements a class for Principal Component Analysis (PCA) on image features. The code includes methods for constructing the EigenImages object, extracting features from an image, training the PCA model on a set of images, and reconstructing images from weight vectors. A GUI window is overlaid on the code, allowing users to adjust the contribution of each of the first 10 Principal Components (PC 0 to PC 9). Each PC has a slider and a numerical value field, all currently set to 0.00. A 'Reset' button is located at the bottom of the GUI.

```
protected EigenImages() {  
}  
  
/**  
 * Construct with the given number of principal components.  
 *  
 * @param numComponents  
 *       the number of PCs  
 */  
public EigenImages(int numComponents) {  
    this.numComponents = numComponents;  
    pca = new FeatureVectorPCA(new ThinSvdPrincipalComponentAnalysis(numComponent  
})  
  
@Override  
public DoubleFV extractFeature(FImage img) {  
    final DoubleFV feature = FImage2DoubleFV.INSTANCE.extractFeature(img);  
  
    return pca.project(feature);  
}  
  
@Override  
public void train(List<? extends FImage> data) {  
    final double[][] features = new double[data.size()][];  
  
    width = data.get(0).width;  
    height = data.get(0).height;  
  
    for (int i = 0; i < features.length; i++)  
        features[i] = FImage2DoubleFV.INSTANCE.extractFeature(data.get(i)).values  
  
    pca.learnBasis(features);  
}  
  
/**  
 * Reconstruct an image from a weight vector  
 *  
 * @param weights  
 *       the weight vector  
 * @return the reconstructed image  
 */  
public FImage reconstruct(DoubleFV weights) {  
    return DoubleFV2FImage.extractFeature(pca.generate(weights), width, height);  
}  
  
/**  
 * Reconstruct an image from a weight vector  
 *  
 * @param weights  
 *       the weight vector  
 * @return the reconstructed image  
 */  
public FImage reconstruct(double[] weights) {  
    return new FImage(ArrayUtils.reshapeFloat(pca.generate(weights), width, height));  
}  
  
/**  
 * Draw a principal component as an image. The image will be normalised so  
 * it can be displayed correctly.  
 *  
 * @param pc  
 *       the index of the PC to draw.  
 * @return an image showing the PC.  
 */  
public FImage visualisePC(int pc) {  
    return new FImage(ArrayUtils.reshapeFloat(pca.getPrincipalComponent(pc), width, height)).normalise();  
}  
  
@Override
```

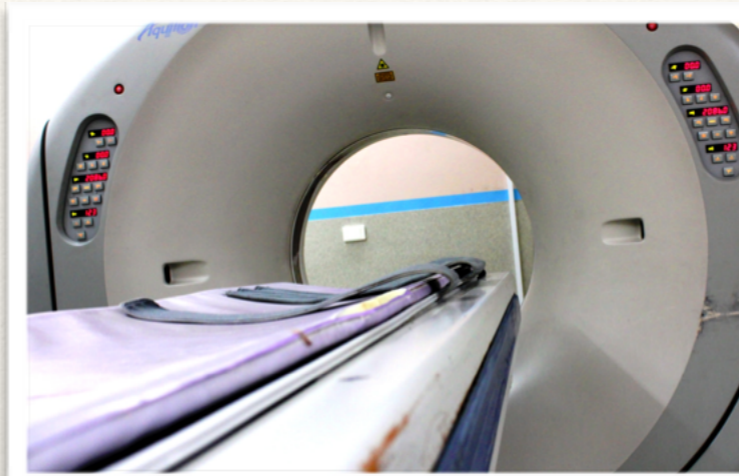
PC 0:  0.00  
PC 1:  0.00  
PC 2:  0.00  
PC 3:  0.00  
PC 4:  0.00  
PC 5:  0.00  
PC 6:  0.00  
PC 7:  0.00  
PC 8:  0.00  
PC 9:  0.00

Reset

Writable Smart Insert 104 : 59



# Image Acquisition Hardware



but how do you go about designing  
a computer vision system? and is  
that all you need?

---

# Key terms in designing CV systems

---

**robust**

**invariant**

**repeatable**

**constraints**



---

# Key terms in designing CV systems

---

**robust**

**invariant**

**repeatable**

*These are what you want*

**constraints**



---

# Key terms in designing CV systems

---

**robust**

**invariant**

**repeatable**

*This is what you design  
your system to be*



---

# Key terms in designing CV systems

---

**robust**

*This is what you apply  
to make it work*

**constraints**



---

# Robustness

---

- ❖ The vision system must be **robust** to changes in its environment
- ❖ i.e. changes in lighting; angle or position of the camera; etc



---

# Repeatability

---

- ❖ **Repeatability** is a *measure* of robustness
- ❖ Repeatability means that the system must work the same over and over, regardless of environmental changes





---

# Invariance

---

- ❖ **Invariance** to environmental factors helps achieve robustness and repeatability
- ❖ Hardware and software can be designed to be invariant to certain environmental changes
  - ❖ e.g. you could design an algorithm to be invariant to illumination changes...



---

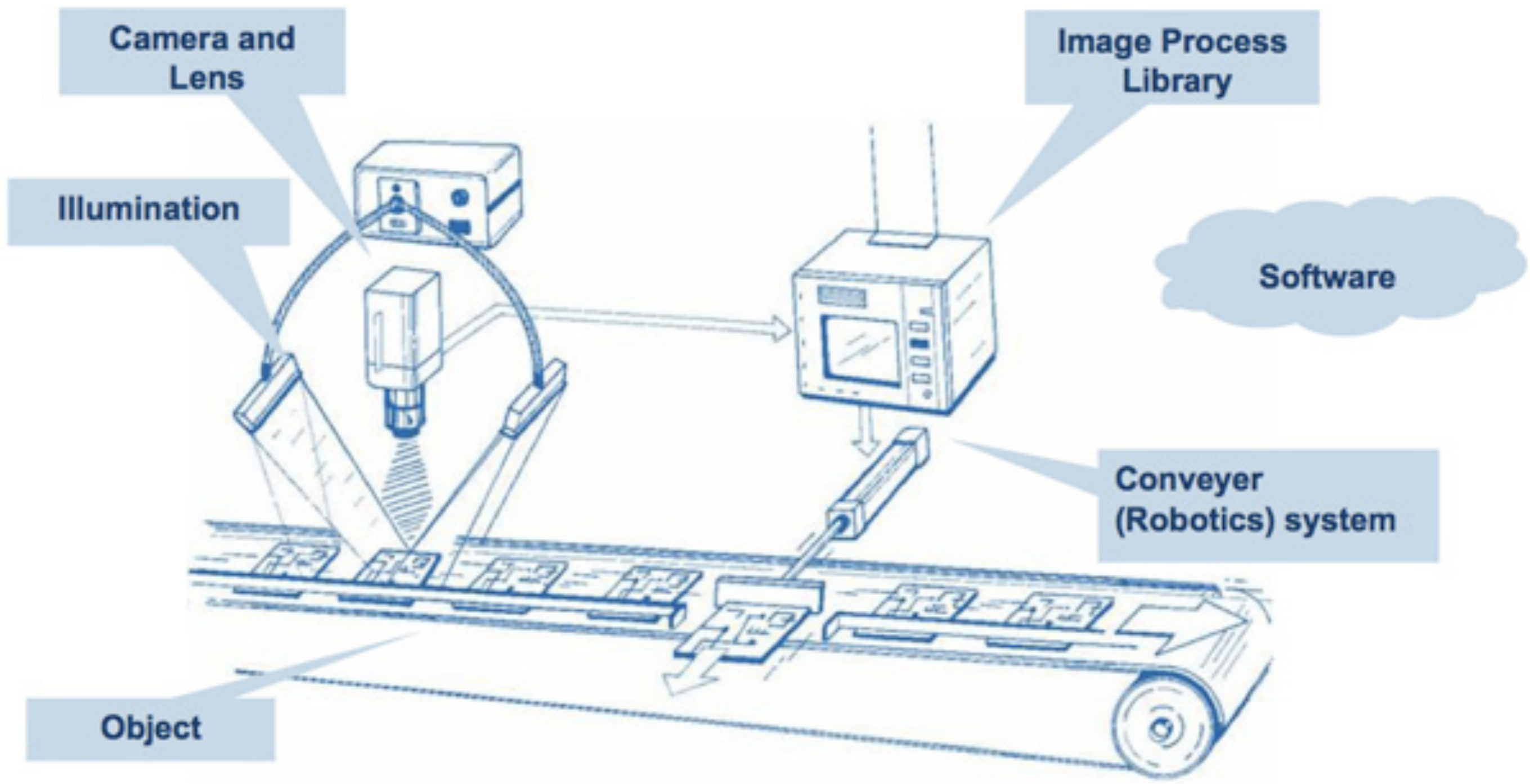
# Constraints

---

- ❖ **Constraints** are what you apply to the hardware, software and wetware to make your computer vision system work in a repeatable, robust fashion.
- ❖ e.g. you constrain the system by putting it in a box so there can't be any illumination changes



# Constraints in Industrial Vision



---

# Software Constraints

---

- ❖ Really simple, but incredibly fast algorithms
- ❖ Hough Transform is popular, but note that it isn't all that robust without physical constraints
  - ❖ Actually, same is true of most algorithms / techniques used in industrial vision
- ❖ Intelligent use of colour...



---

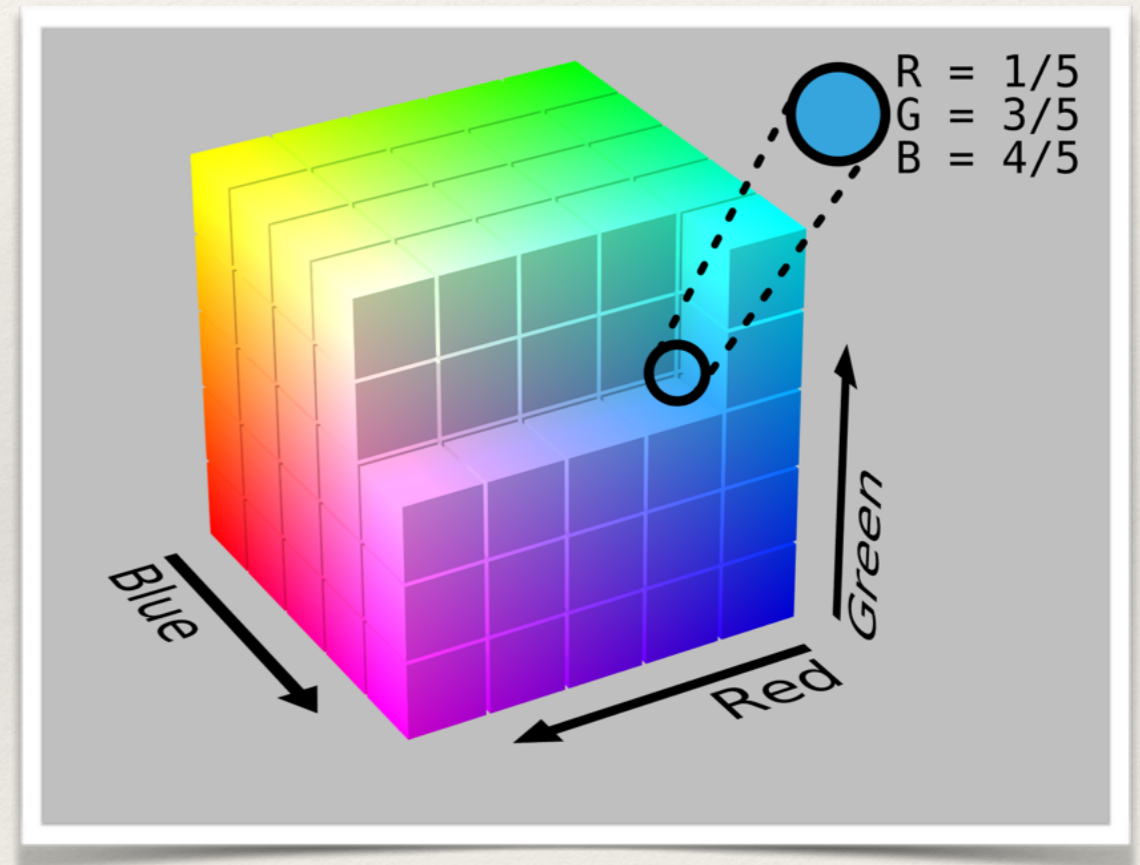
# *Important aside:* Colour-spaces

---

- ❖ There are many different ways of *numerically* representing colour
- ❖ A single representation of all possible colours is called a colour-space
- ❖ It's *generally* possible to convert to one colour-space to another by applying a mapping (in the form of a set of equations or an algorithm)

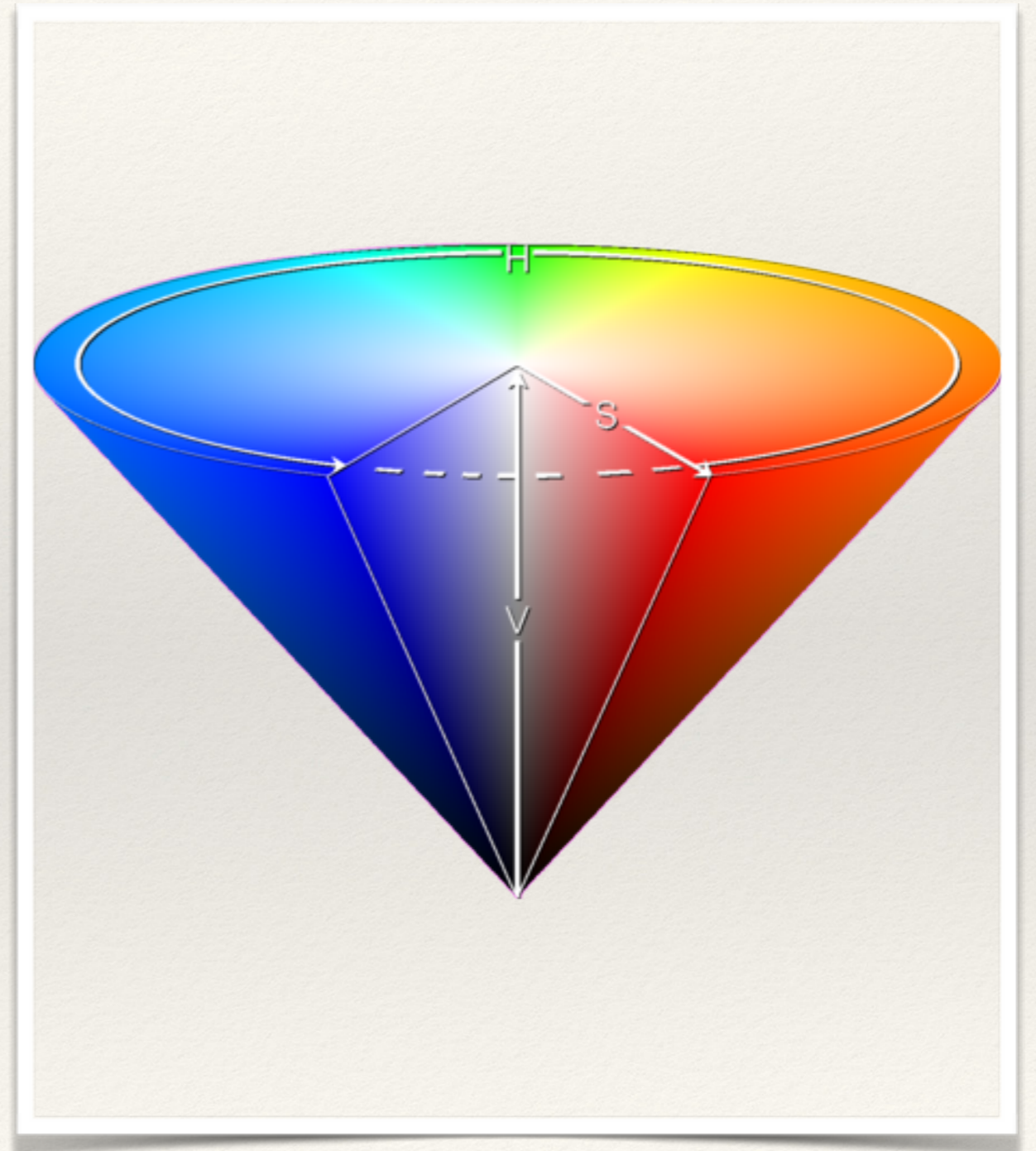
# RGB Colour-space

- ❖ Most physical image sensors capture RGB
- ❖ By far the most widely known space
- ❖ RGB “couples” brightness (luminance) with each channel, meaning that illumination invariance is difficult.



# HSV Colour-space

- ❖ Hue, Saturation, Value is another colour-space
  - ❖ Hue encodes the pure colour as an angle
    - ❖ **red == 0° == 360° !!**
  - ❖ Saturation is how vibrant the colour is
  - ❖ And the Value encodes brightness
  - ❖ A simple way of achieving invariance to lighting is to use just the H or H & S components





*Demo: colour-spaces*

---

# Physical Constraints

---

- ❖ Industrial vision is usually solved by applying simple computer vision algorithms, and lots of physical constraints:
- ❖ Environment: lighting, enclosure, mounting
- ❖ Acquisition hardware: expensive camera, optics, filters



*Let's look at some types of physical  
constraint*

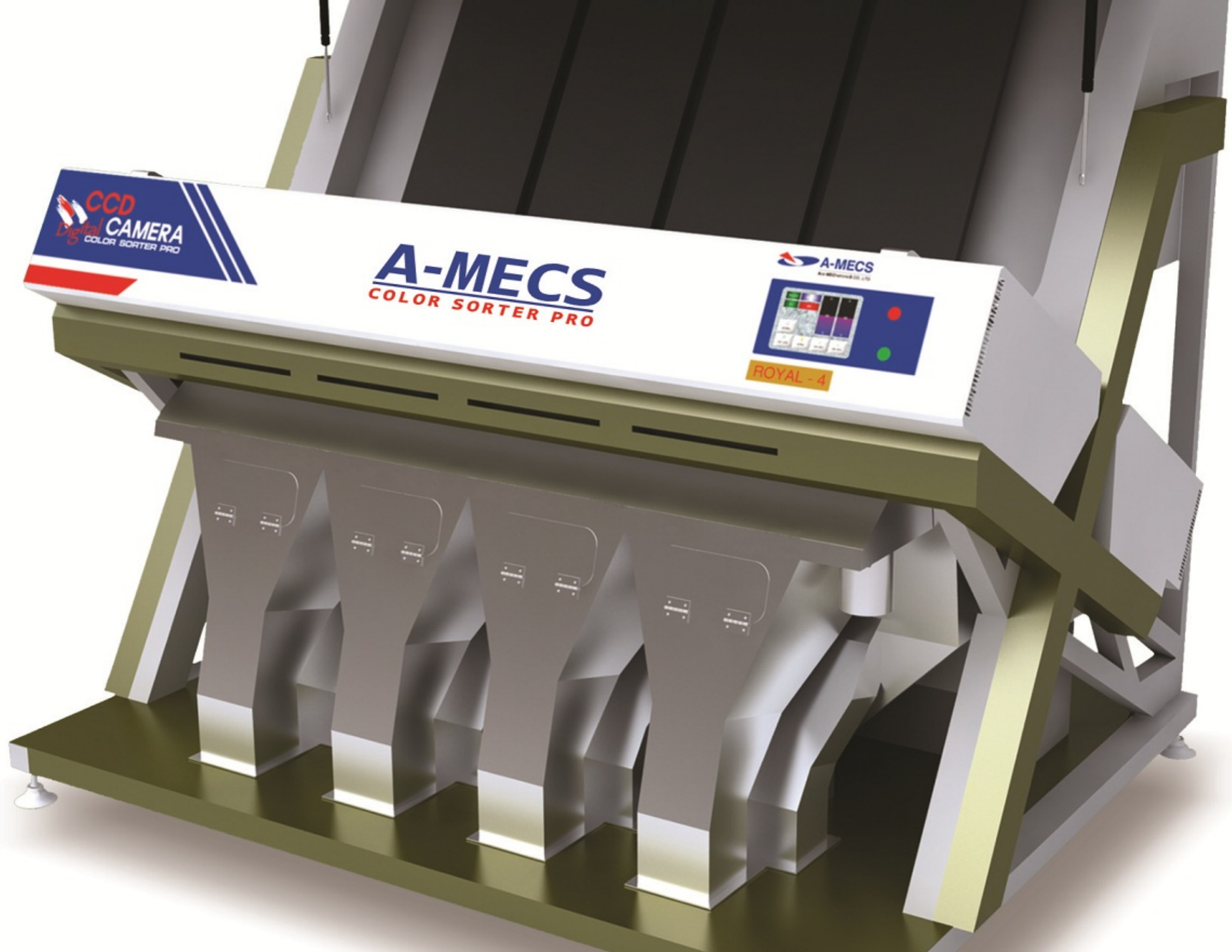
CCD  
Digital CAMERA  
COLOR SORTER PRO

**A-MECS**  
COLOR SORTER PRO

A-MECS  
AG-MECS@GMAIL.COM



ROYAL - 4





---

# Vision in the wild

---

- ❖ So, what about vision systems in the wild, like ANPR cameras, or recognition apps for mobile phones?
- ❖ Apply as many hardware and wetware constraints as possible, and let the software take up the slack
- ❖ Colour information often less important than luminance



# ANPR constraints

- ❖ License plate styles are different across the world, so most ANPR systems will only work with plates from a single country.
- ❖ License plates themselves are constrained in design:
  - ❖ Dimensions
  - ❖ Font
  - ❖ Material (IR reflectance!)

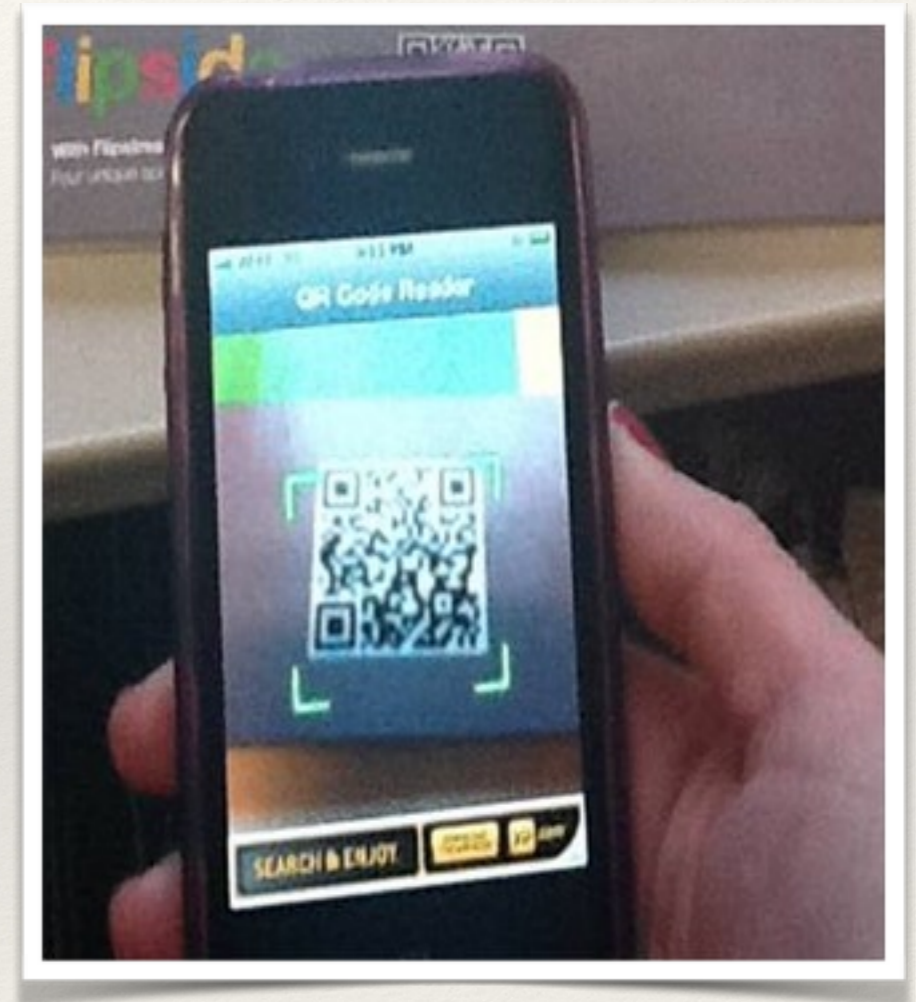


---

# Mobile vision constraints

---

- ❖ QR-Codes are designed to be robust
- ❖ But most software requires (constrains) the user to operate in a certain way:
  - ❖ Orientation - approximately upright
  - ❖ Within a certain area
  - ❖ Approximately stationary





---

# Almost unconstrained vision?

---

- ❖ As computers become more powerful, and new software techniques are developed to deal with invariance the need for constraints becomes less.
- ❖ ...but there is always going to be a problem of optimising the costs, and constraints can always help reduce costs

---

# Summary

---

- ❖ **Robust and repeatable** computer vision is achieved through engineered **invariance** and applied **constraints**.