

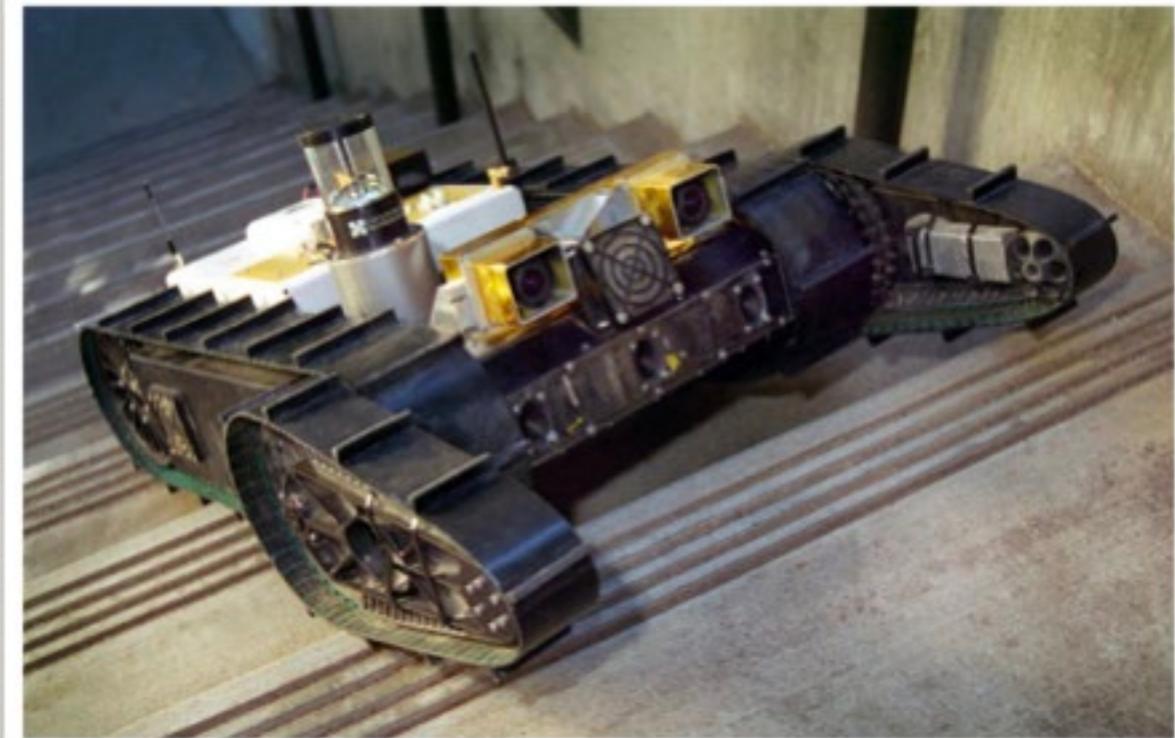
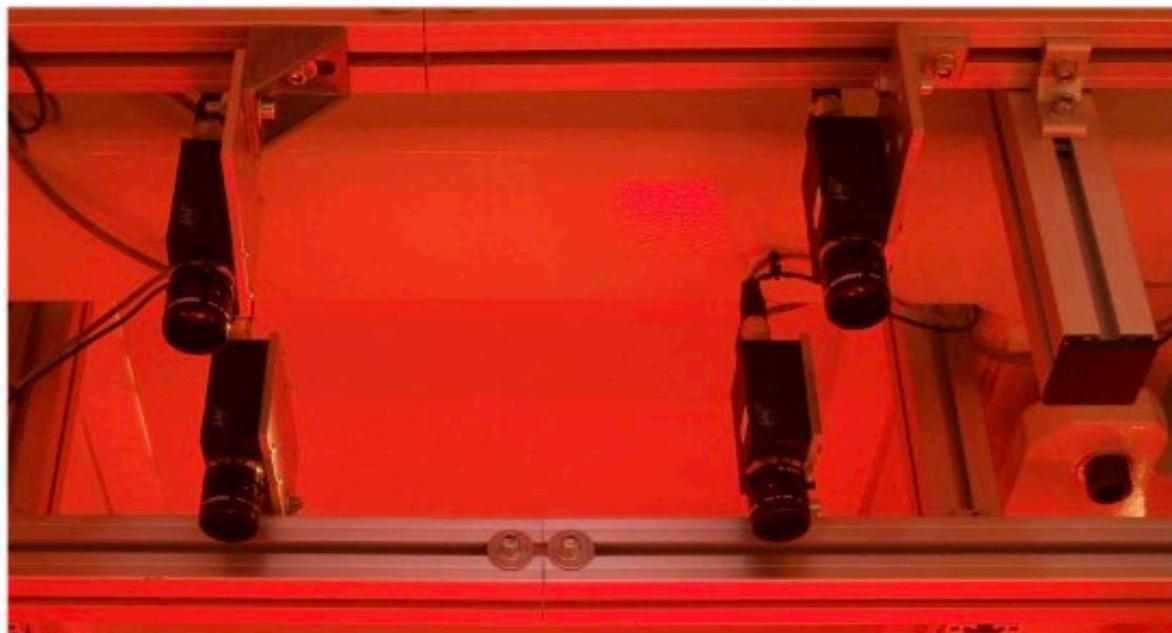
*Computer Vision*

---

# Building machines that see

Hansung Kim  
[h.kim@soton.ac.uk](mailto:h.kim@soton.ac.uk)

# Types of Computer Vision and their Environment



Scanning.

www.metmuseum.org/Cc

THE METROPOLITAN MUSEUM OF ART

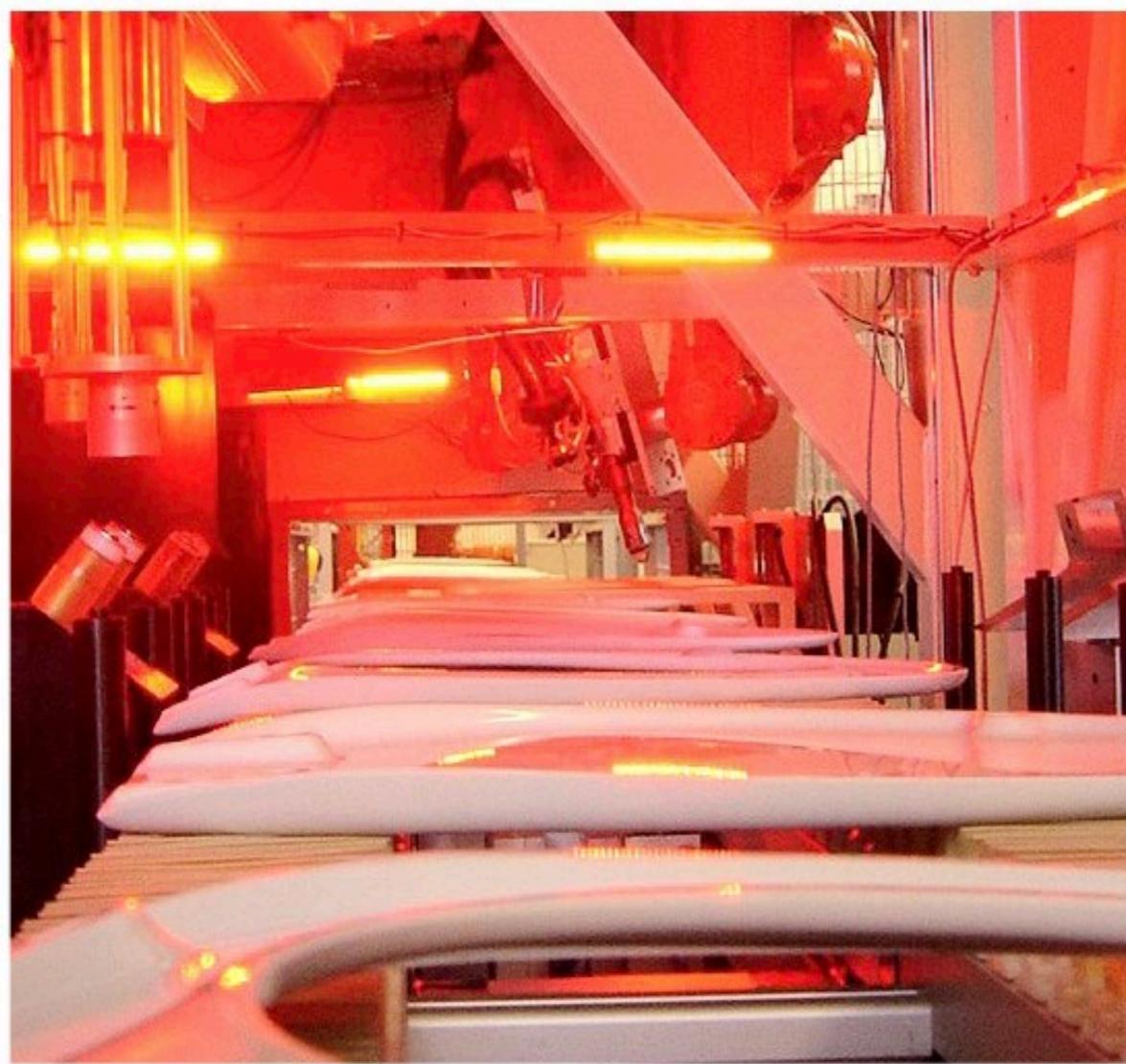
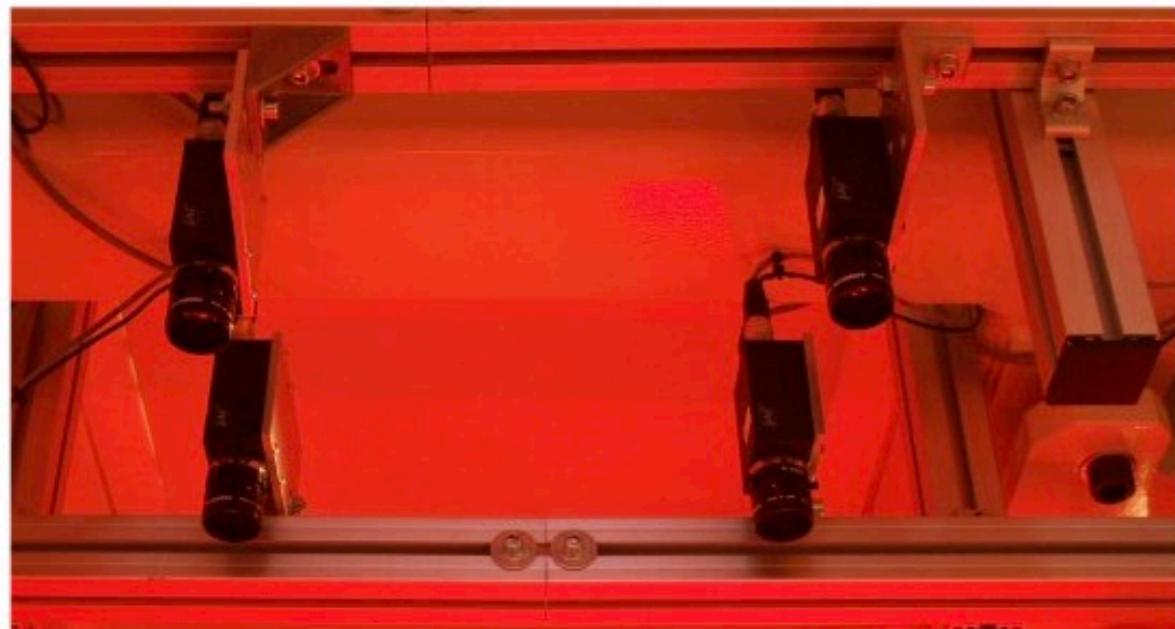
Madame X (Madame Pierre Gautreau)  
John Singer Sargent (American; Florence 1856–1925 London)

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



# Industrial Vision

Scanning.

www.metmuseum.org/Cc

M THE METROPOLITAN MUSEUM OF ART

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

Enlarge Image

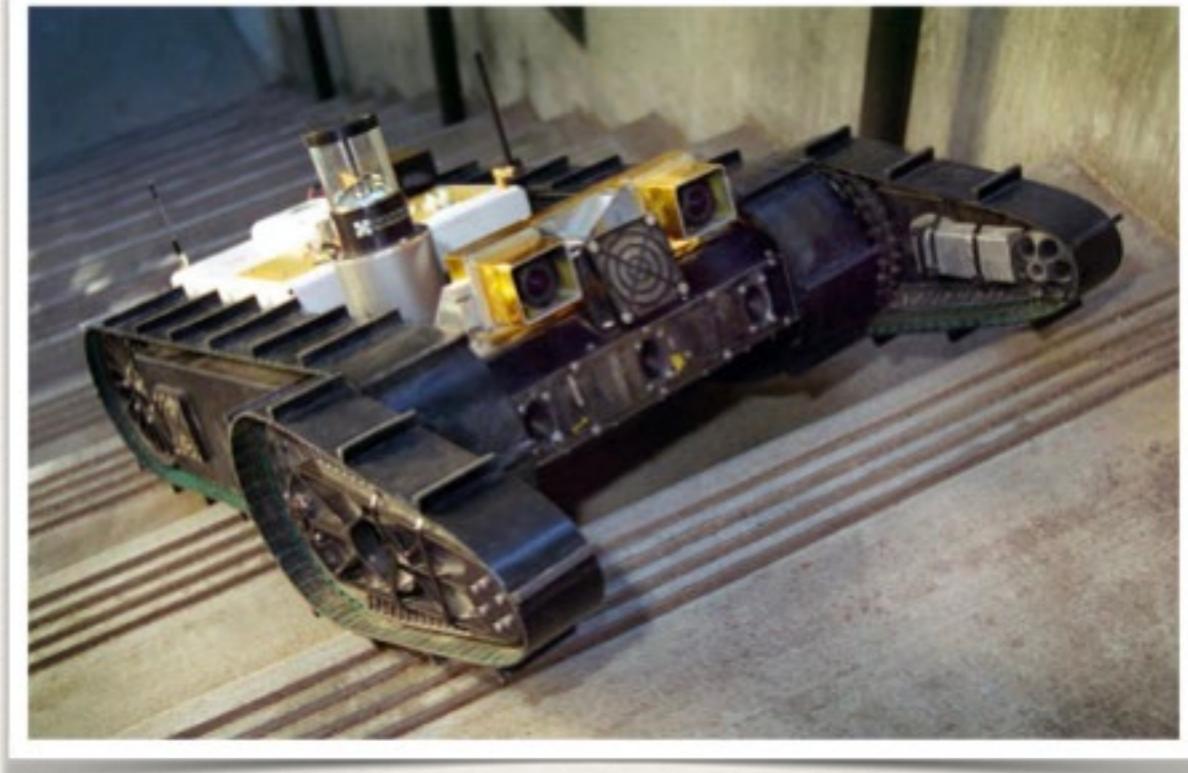
Date: 1883–84  
Medium: Oil on canvas  
Dimensions: 82 1/8 x 43 1/4 in. (208.6 x 109.9 cm)  
Classification: Paintings  
Credit Line: Arthur Hoppock Hearn Fund, 1916  
Accession Number: 16.53  
This artwork is not on display

+ Description

+ Signatures, Inscriptions, and



# Robot Vision



A screenshot of a mobile device displaying a webpage from the Metropolitan Museum of Art. The main image on the screen is a painting titled "Madame X" by John Singer Sargent. The painting depicts a woman in a dark blue velvet dress, shown from the waist up, facing slightly to her left. The device's status bar shows the time as 1:41. The browser address bar shows the URL www.metmuseum.org/Cc. The page includes the Met logo and the text "THE METROPOLITAN MUSEUM OF ART". Below the painting, there is descriptive text: "Madame X (Madame Pierre Gautreau) John Singer Sargent (American, Florence 1856–1925 London)". There are also links for "Enlarge Image", "Description", and "Signatures, Inscriptions, and". The bottom of the screen shows standard mobile navigation icons.



# Vision in the wild



Scanning.

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 computer vision  
systems have in common?

# Cameras



VS.



Samsung Galaxy S10 5G

Released in 2017

12 Mpixels

£690

Canon G7X

Released in 2014

12 Mpixels

£400

# Cameras



(1)



(2)

# Cameras



Samsung Galaxy



Canon G7X

700% Zoomed-In

# Cameras



(1)



(2)

# Cameras



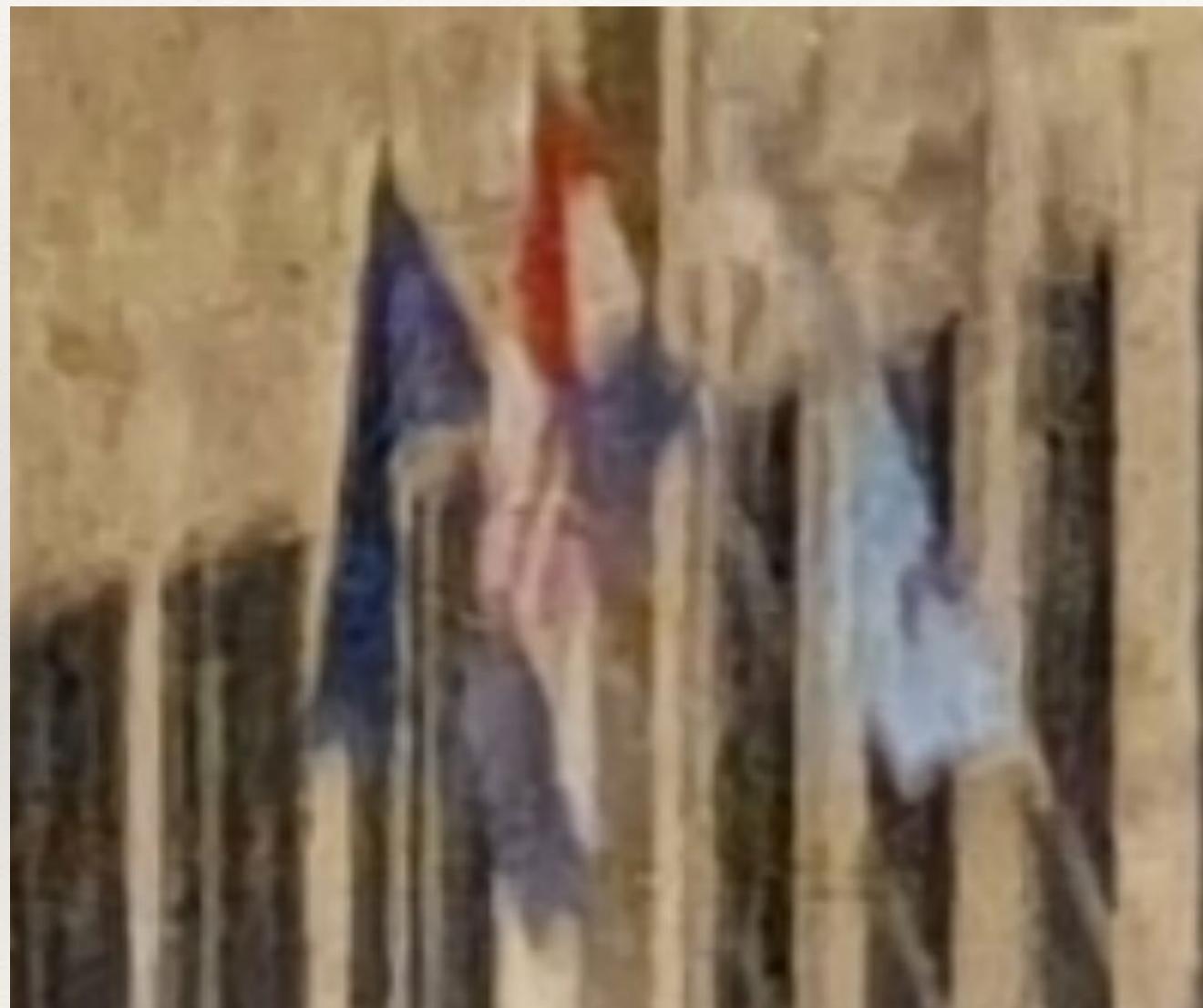
Samsung Galaxy



Canon G7X

100% Zoomed-In

# Cameras



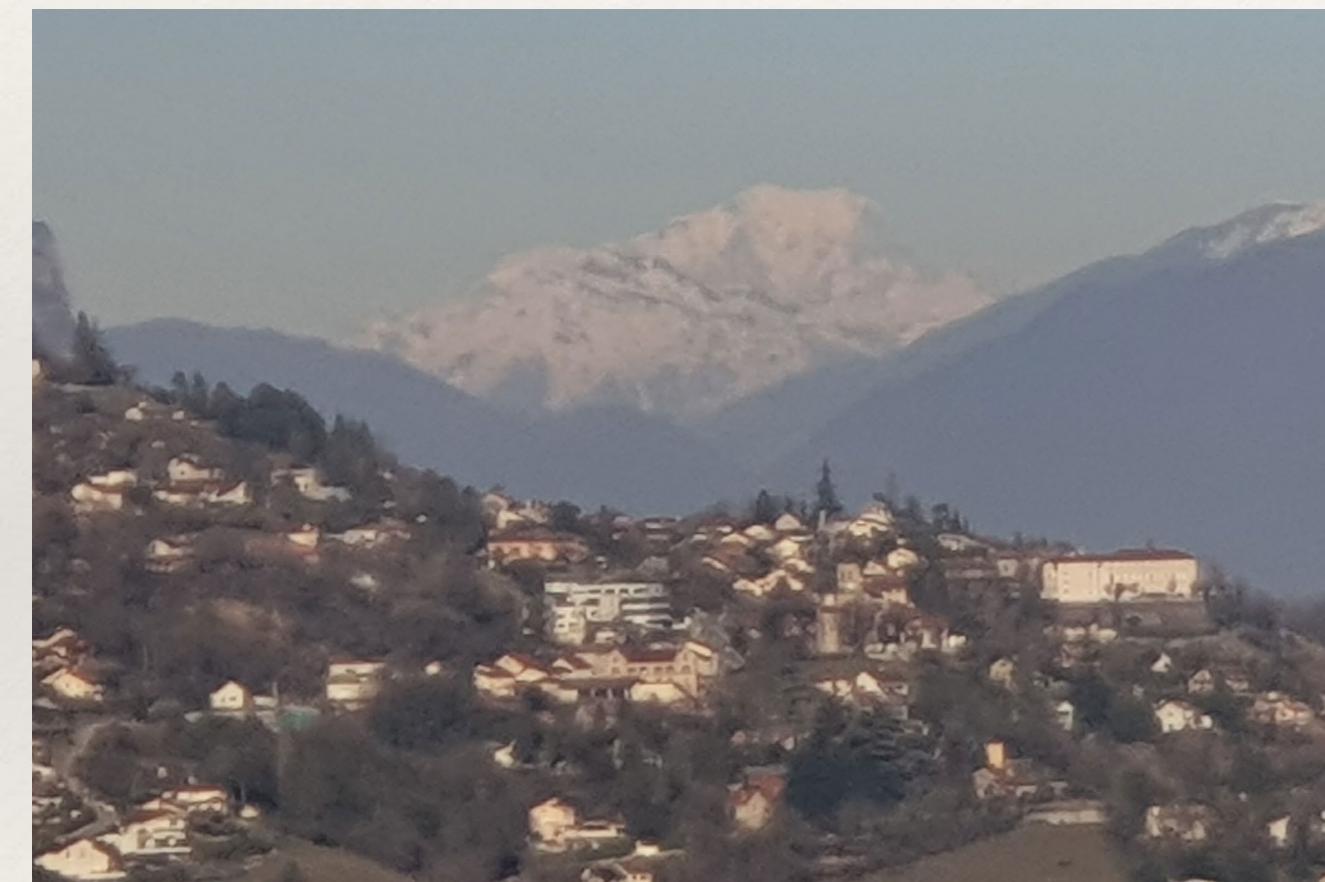
Samsung Galaxy



Canon G7X

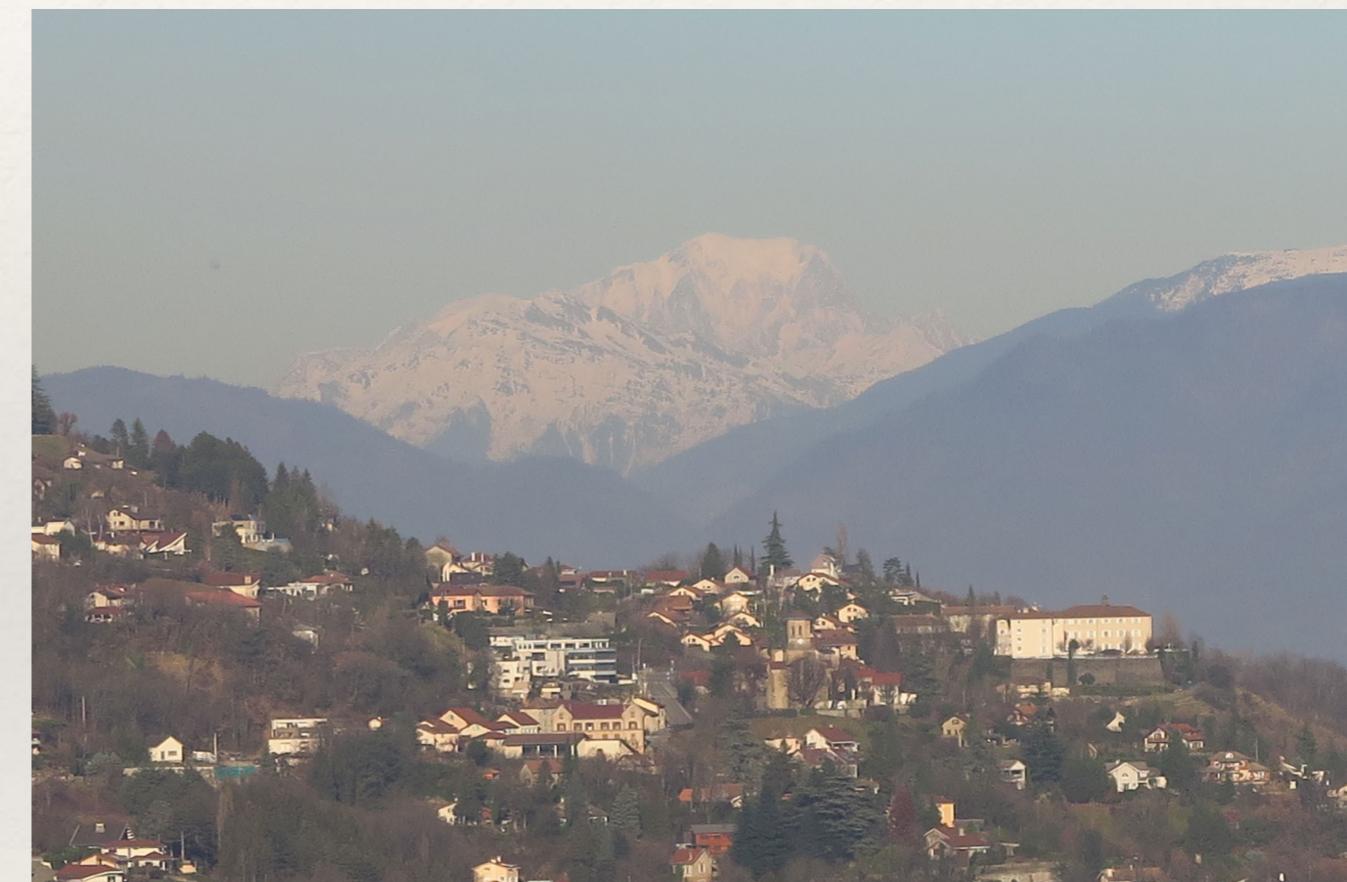
500% Zoomed-In

# Cameras



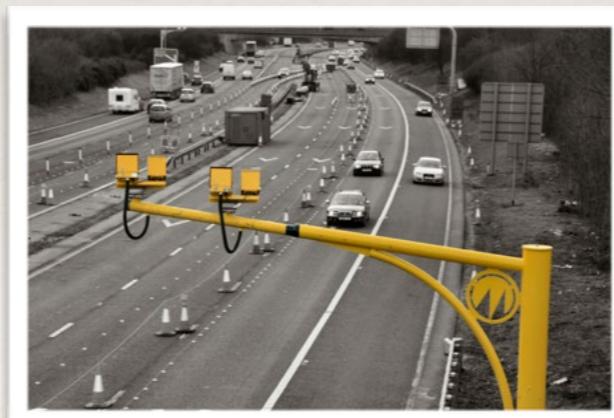
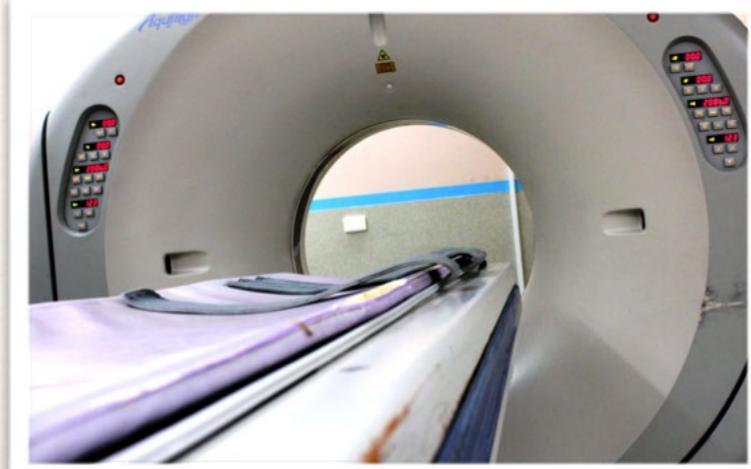
Digital zoom by Galaxy

Zoom-In Quality



Optical zoom by G7X

# Image Acquisition Hardware



# Computer Vision Software

The screenshot shows the Eclipse IDE interface with two main windows. On the left, the code editor displays the `EigenImages.java` file, which contains Java code for a Principal Component Analysis (PCA) implementation. The code includes methods for extracting features from images, training the model with a list of images, and reconstructing images from weight vectors. On the right, a graphical user interface (GUI) window titled "EigenImages" is displayed. This window has a title bar with "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", and "PC 9: 0.00". Below these sliders is a "Reset" button. The background of the GUI window is dark gray.

```
Java - image-feature-extraction/src/main/java/org/openimaj/image/model/EigenImages.java - Eclipse SDK - /Users/jsh2/Documents/LMLK-Workspace
```

```
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(numComponents));
}

@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();
}
```

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

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

invariant

robust

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

is *This is what you apply* to make it work

constraints



# Robustness

- ❖ The vision system must be **robust** to changes in its environment
  - ❖ i.e. lighting changes; sensor noise; defocusing; position of 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 scale, rotation, illumination changes...

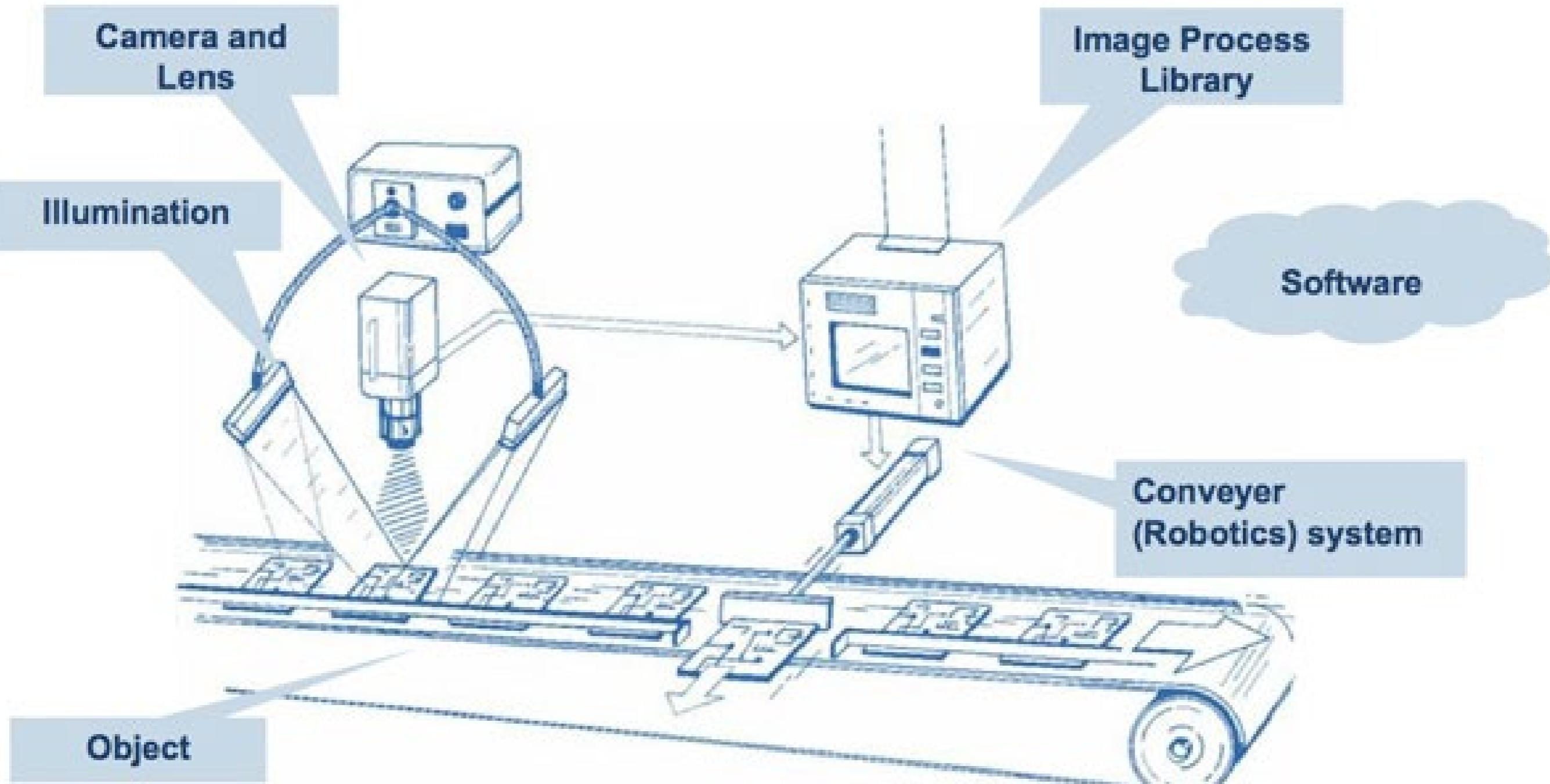


# Constraints

- ❖ **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

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



---

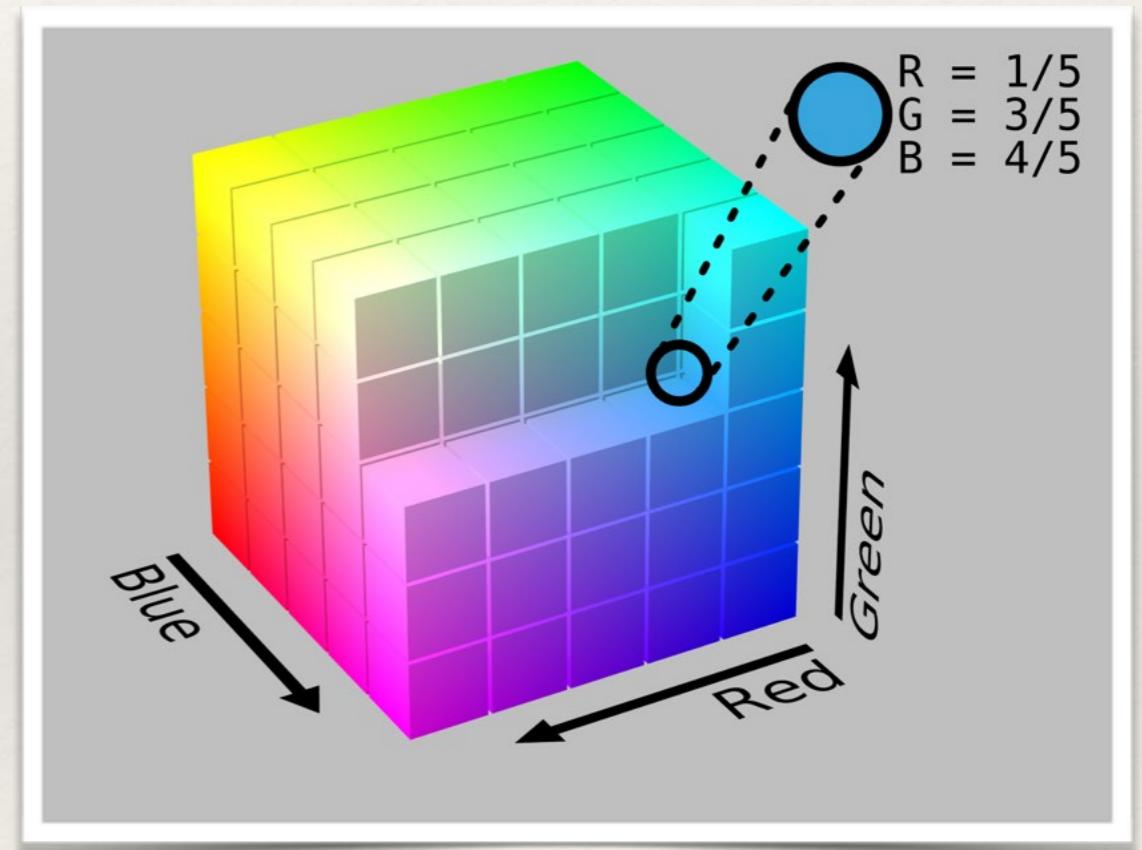
# Important aside: *Colour-spaces*

---

- ❖ 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)

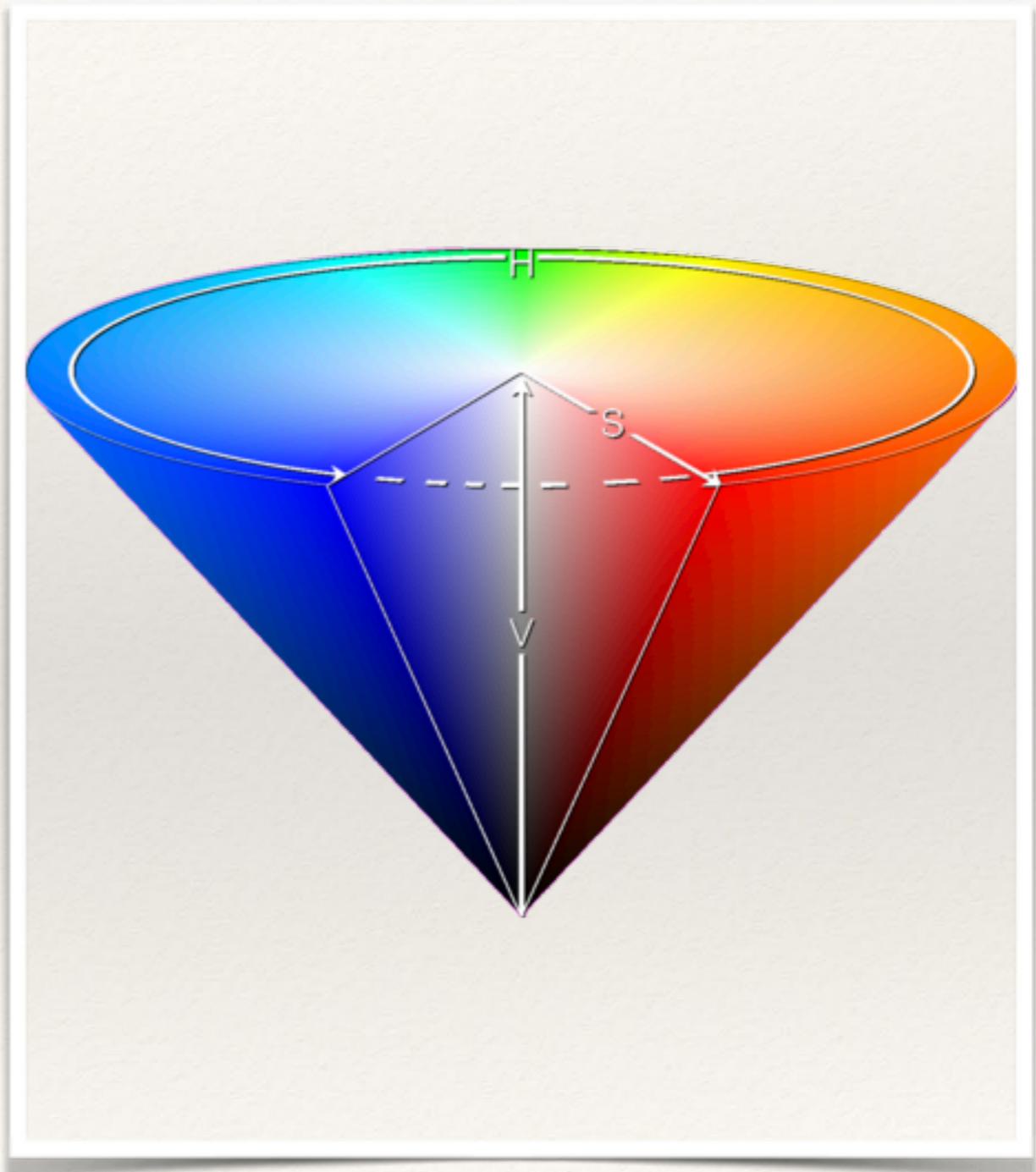
# RGB Colour-space

- ❖ 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 ==  $0^\circ == 360^\circ$  !!**
  - ❖ 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

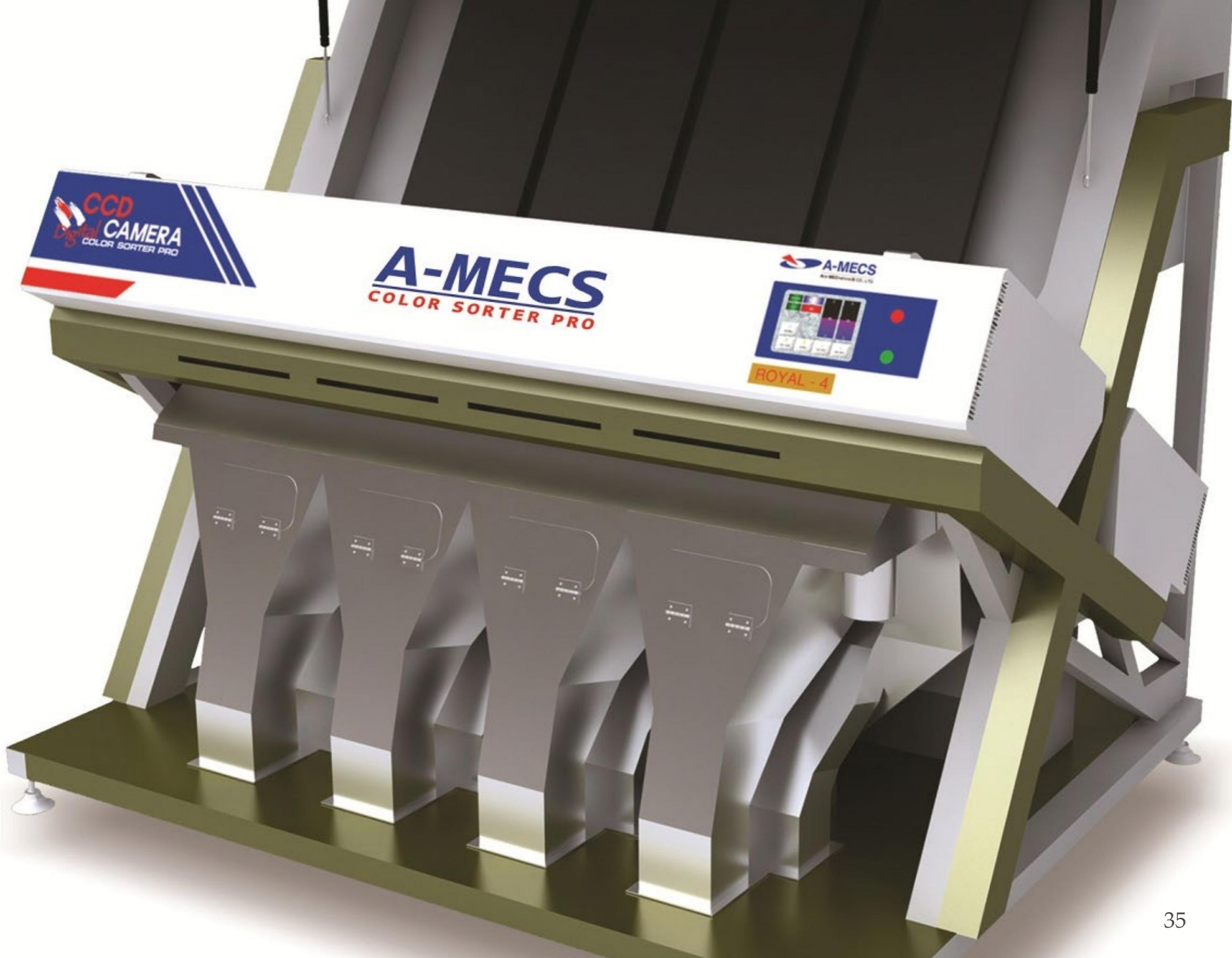
- ❖ Industrial vision is usually solved by applying simple computer vision algorithms, and lots of physical constraints:
  - ❖ Environment: lighting, enclosure, mounting
  - ❖ Acquisition hardware: expensive camera, optics, filters



# Physical Constraints

- ❖ Some types of hardware constraints
  - ❖ Different lenses (Focal length, Sharpness, Aperture)
  - ❖ Different sensor (Infra Red)
  - ❖ Different filters (Polarizing, Neutral Density, Colour filters, etc)







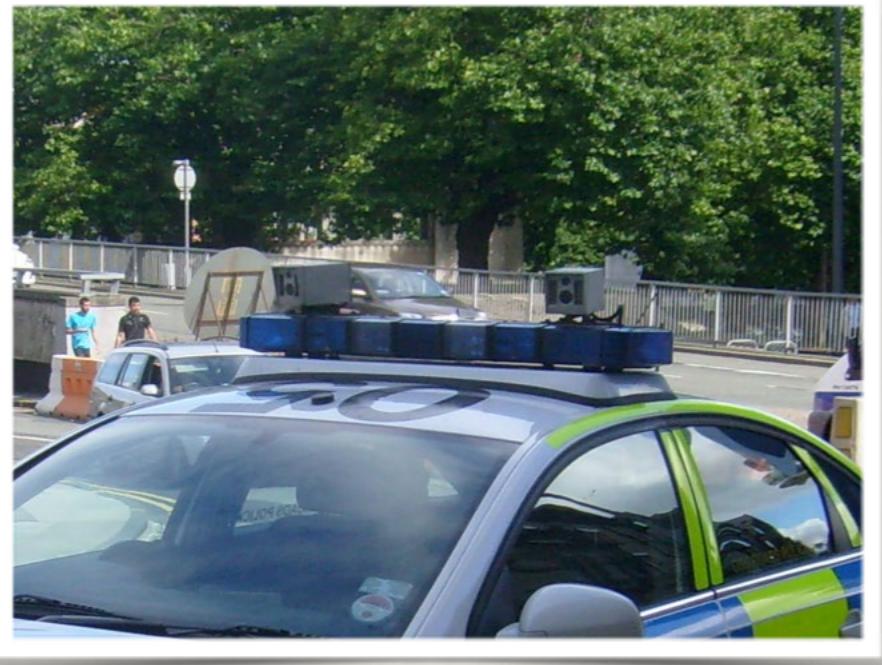
# Vision in the wild

- ❖ 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



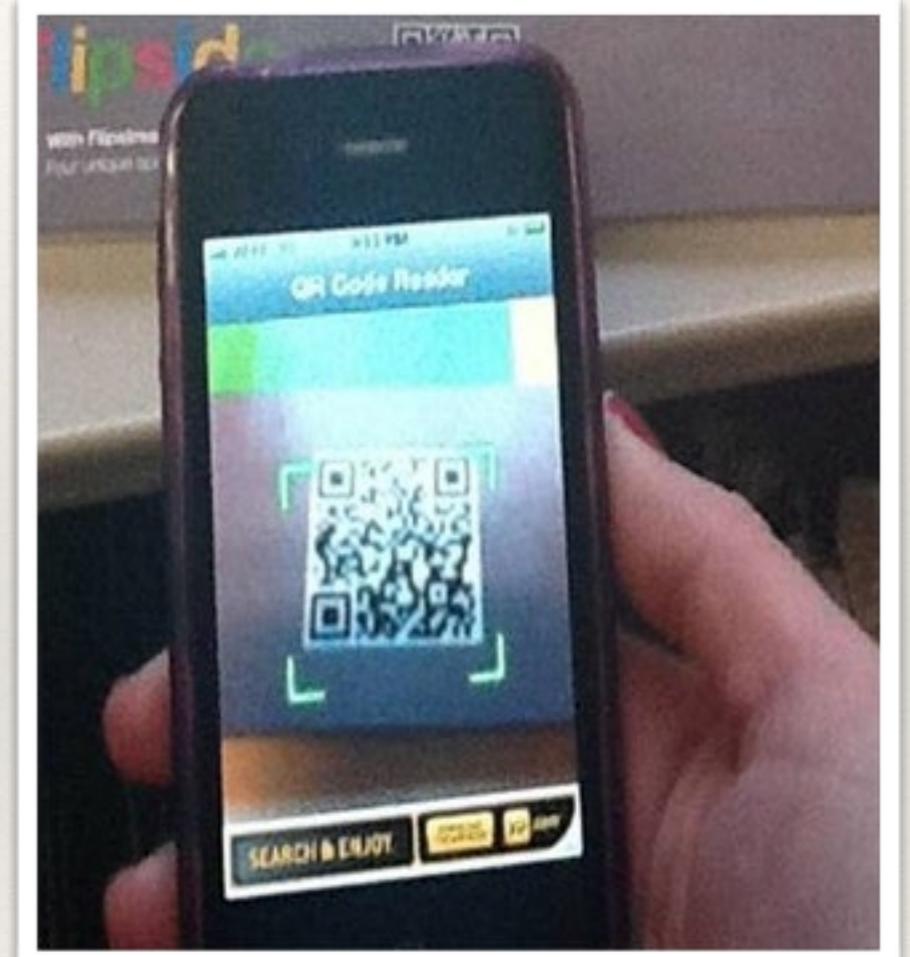
# ANPR constraints

- ❖ 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

---

- ❖ **Summary**
  - ❖ Complete CV system: hardware + software + wetware
- ❖ **System design**
  - ❖ **Robust** and **repeatable** computer vision is achieved through engineered **invariance** and applied **constraints**.
- ❖ **Colour Space:** RGB vs HSV

# Further reading and exercises

---

## ❖ Further reading

- White papers on lighting for industrial vision:  
<http://www.ni.com/white-paper/6901/en/>
- Further information on HSV (and some alternative) colour-spaces, including code (C++):  
<http://archive.is/NEzmQ>
- Mark's Book (third edition): Appendix 4 covers colour images and colour-spaces

# Further reading and exercises

---

## ❖ Practical exercises

- Java - OpenIMAJ tutorial

<http://www.openimaj.org/tutorial/>

Play with the org.openimaj.image.colour.ColourSpace class

- Python Imaging Library is a good library

<https://pypi.org/project/Pillow/>

- OpenCV-Python for more advanced

[https://docs.opencv.org/4.5.5/da/df6/tutorial\\_py\\_table\\_of\\_contents\\_setup.html](https://docs.opencv.org/4.5.5/da/df6/tutorial_py_table_of_contents_setup.html)

- Explore different colour-spaces

[https://docs.opencv.org/4.5.5/df/d9d/tutorial\\_py\\_colorspaces.html](https://docs.opencv.org/4.5.5/df/d9d/tutorial_py_colorspaces.html)