MTRN4110 Robot Design
Week 7 – Vision I

Liao “Leo” Wu, Lecturer

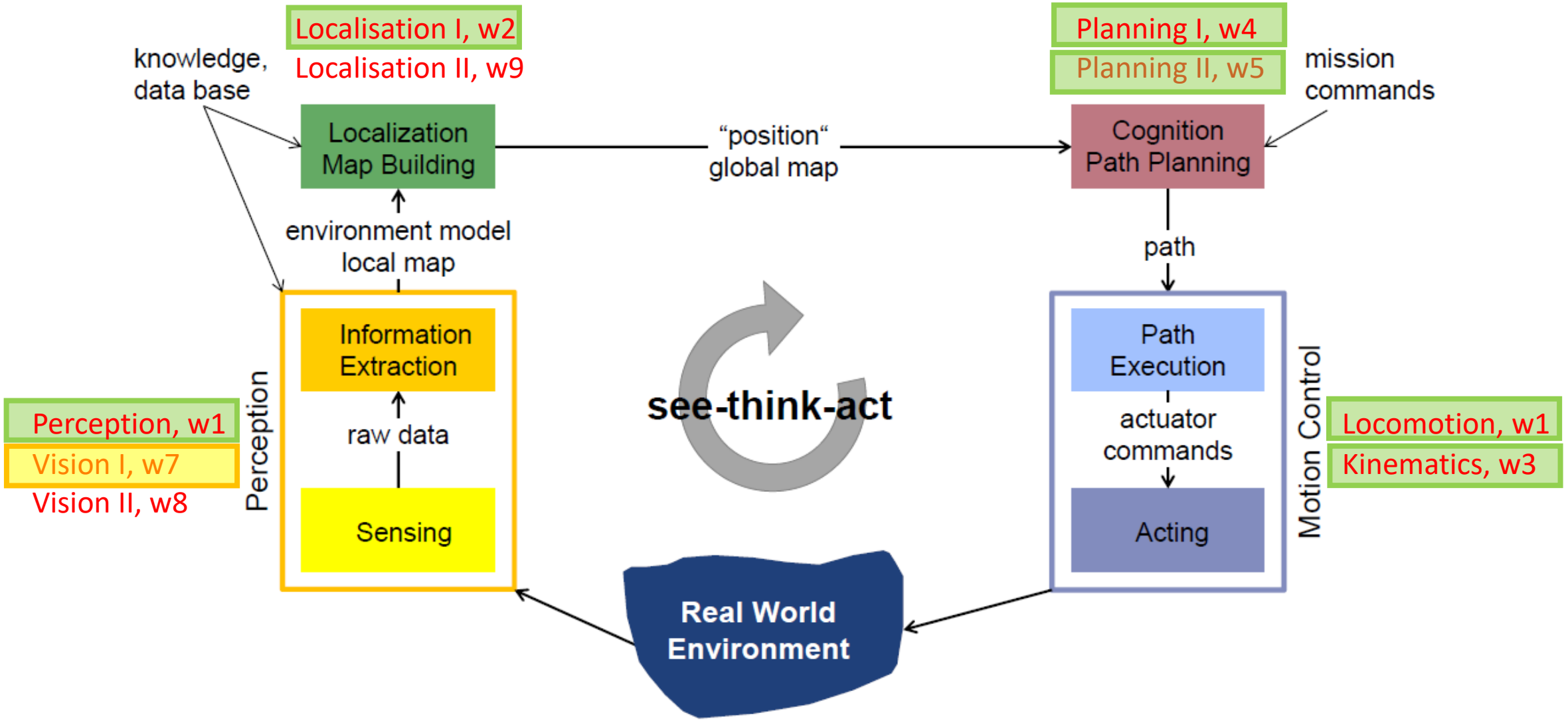
School of Mechanical and Manufacturing Engineering

University of New South Wales, Sydney, Australia

https://sites.google.com/site/wuliaothu/

UNSW SYDNEY

The See-Think-Act cycle



R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.

Today's Agenda

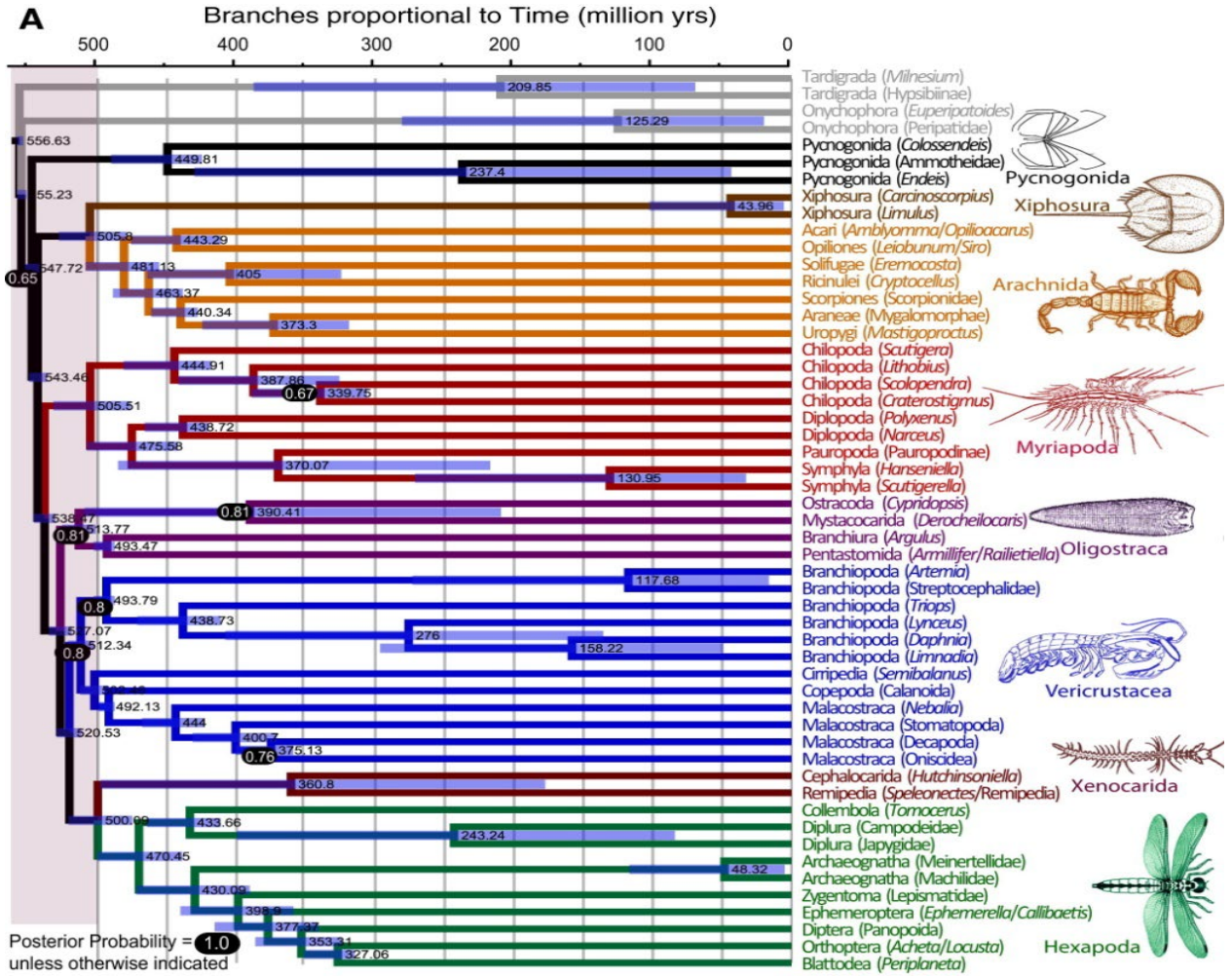
- Introduction to computer vision
  - Connection with image processing and machine learning
  - Tools available

- Image processing
  - Pixels
  - Perspective transforms
  - Colour spaces
  - Thresholding
  - Morphological operations

Introduction to Computer Vision

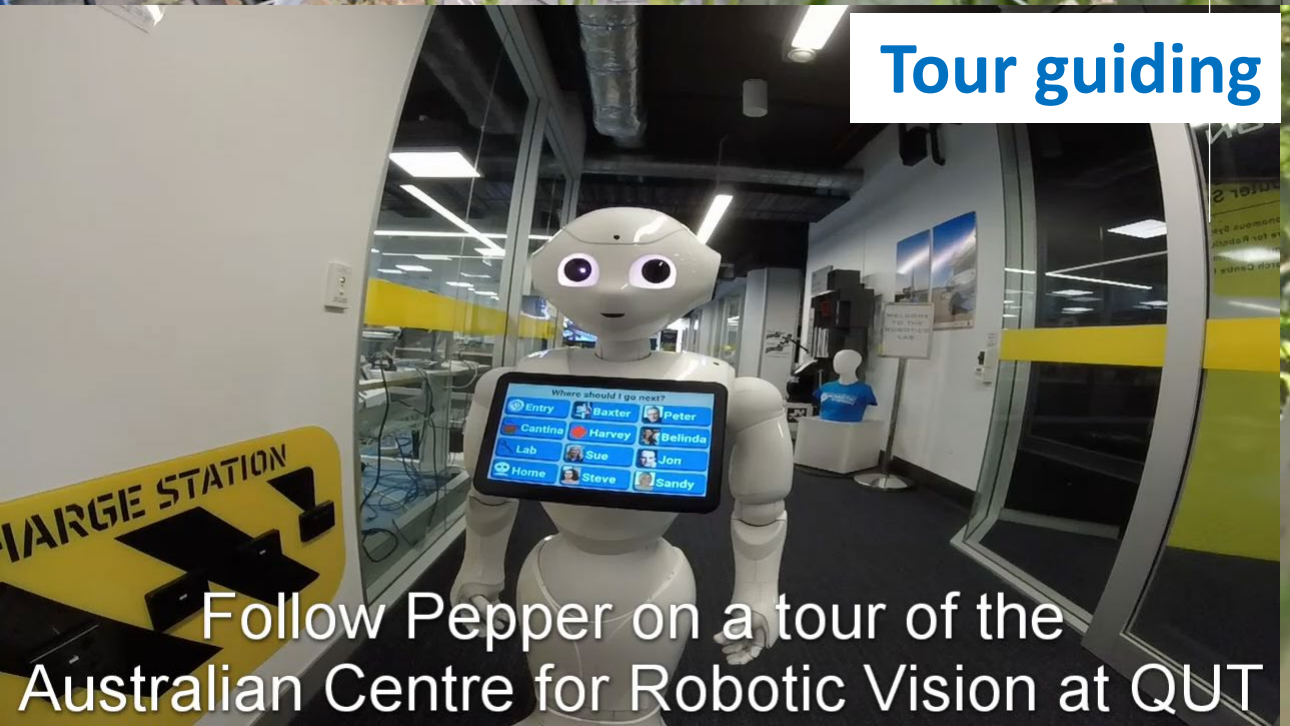
Why vision? – A perspective from nature evolution

- Cambrian Explosion - Most important evolutionary event in the history of life on Earth



Fruit harvesting

Great Barrier Reef Protection

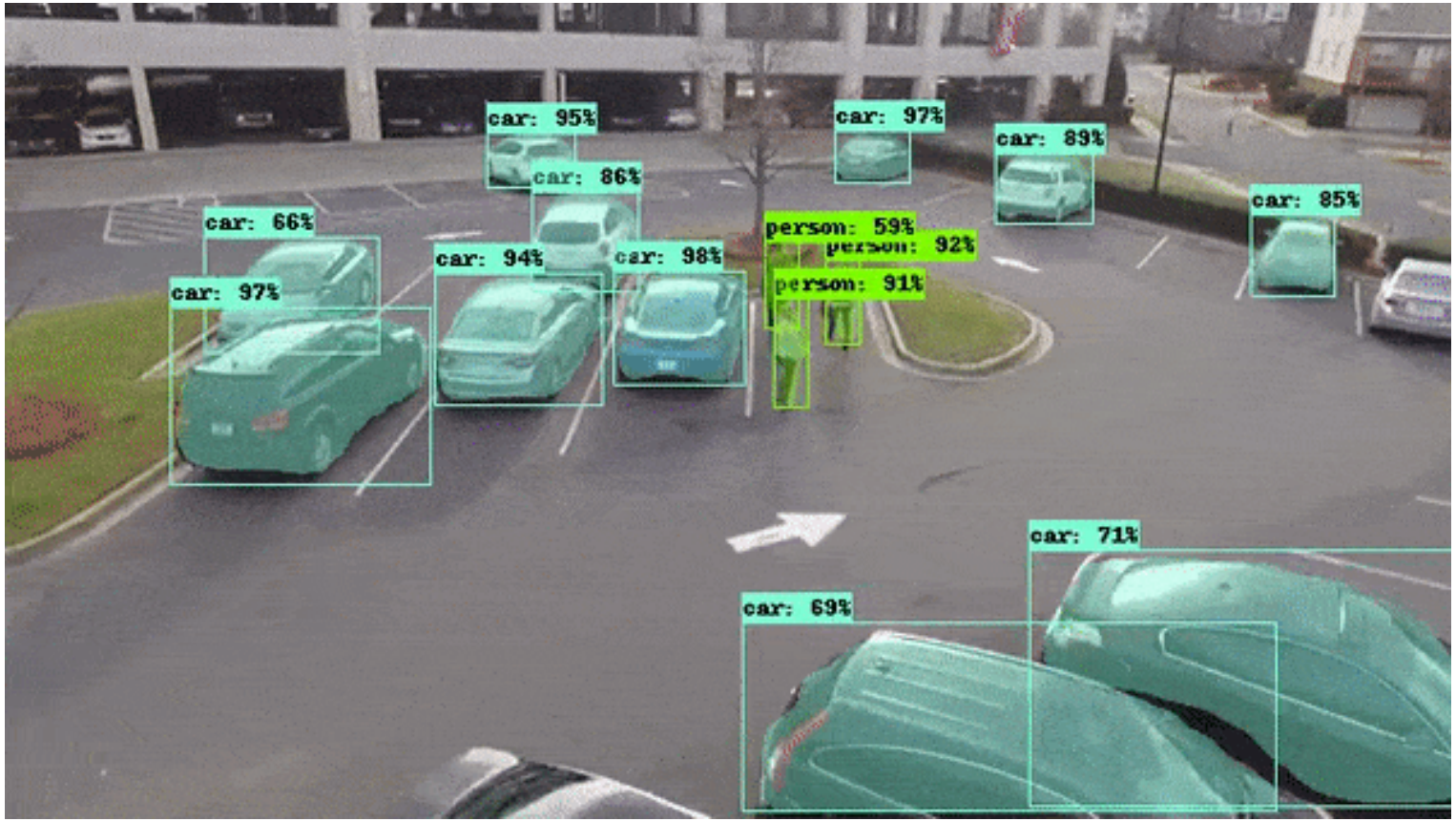


Tour guiding

Koala counting

Computer vision

An interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos.



https://en.wikipedia.org/wiki/Computer_vision
https://www.kdnuggets.com/2019/02/everything-computer-vision.html

What is the difference between Computer Vision and Image Processing?

Input \ Output	IMAGE	INFORMATION
IMAGE	Image Processing	Computer Vision
INFORMATION	Computer Graphics	Artificial Intelligence (not vision-based)

- The line between Image Processing and computer vision is blur.
- Computer vision usually takes image processing as the first step, and then apply machine learning techniques for higher-level tasks.
- Recent progress in "End-to-End" training in computer vision enables to skip the image processing step.
- Nevertheless, image processing is still useful to improve accuracy, efficiency, etc.

https://www.geeksforgeeks.org/digital-image-processing-basics/

Tools Available

Some tools for learning and implementing Computer Vision

https://www.learnopencv.com/opencv-c-vs-python-vs-matlab-for-computer-vision/

OpenCV + Python

- OpenCV
  - An open source computer vision and machine learning software library
  - Has more than 2500 optimised algorithms
  - Multi-language: Has C++, Python, Java and MATLAB interfaces
  - Cross-platform: Supports Windows, Linux, Android and macOS
  - Aims at real-time computer vision applications

- Python
  - An interpreted, high-level, general-purpose programming language
  - Very easy to learn for beginners
  - Top 1 In-Demand programming language to learn in 2021

https://opencv.org/about/
https://www.northeastern.edu/graduate/blog/most-popular-programming-languages/

Comparison of tools for learning and implementing Computer Vision

MATLAB	OpenCV + C++	OpenCV + Python

• Pros
  • Powerful matrix library
  • Toolboxes
  • Visualization and debugging tools
  • Great documentation
  • Large research community
	• Pros
	  • Free!
	  • Huge optimized library
	  • Fast runtime
	  • Platforms and devices
	  • Big community
		• Pros
		  • Ease of use (very friendly for beginners)
		  • Popularity of Python for scientific computing
		  • Visualization and debugging

• Cons
  • Not free (expensive)
  • Slower runtime
  • Difficulty in deployment with embedded hardware
	• Cons
	  • Difficult for beginners
	  • Weak documentation
	  • Visualization and debugging
	  • Small machine learning library
		• Cons
		  • Weak documentation
		  • Lack of support
		  • Slower runtime than C++
		  • OpenCV is written in C/C++

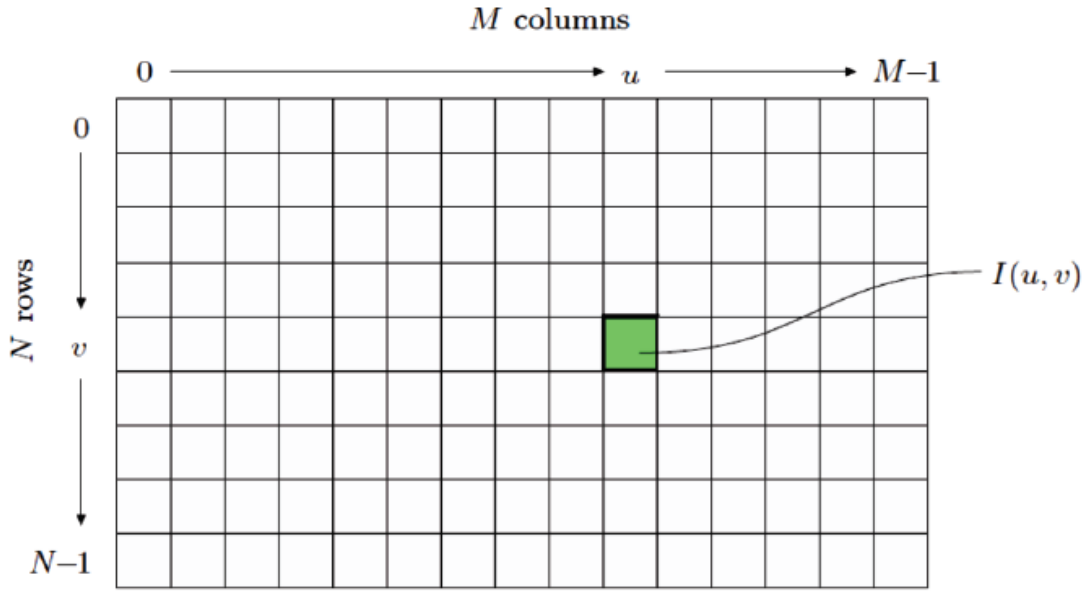
https://www.learnopencv.com/opencv-c-vs-python-vs-matlab-for-computer-vision/

Image Processing

Pixels

How is an image represented digitally?

What we see

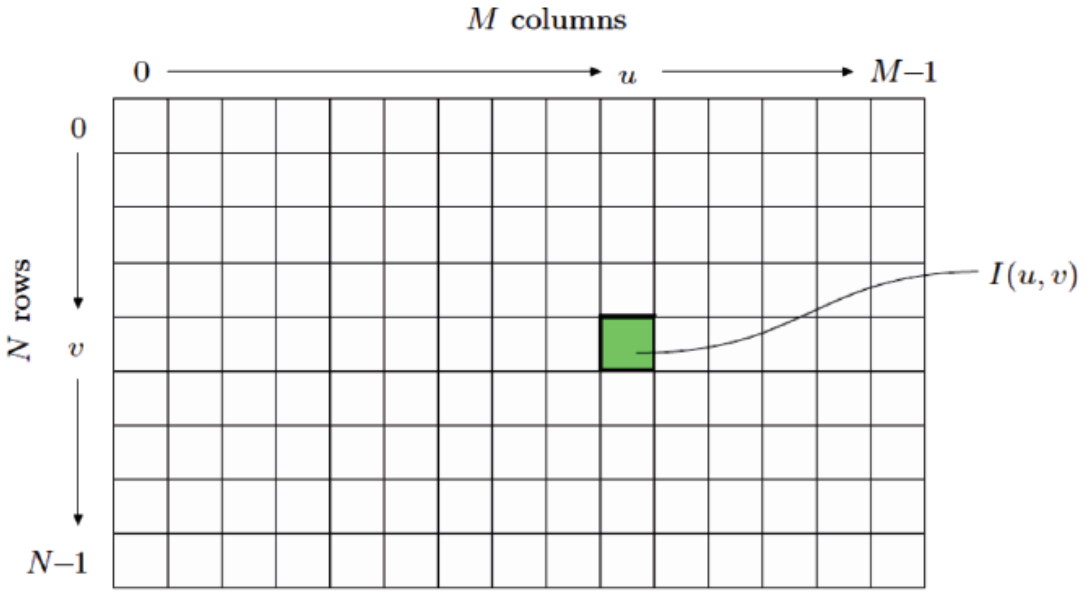


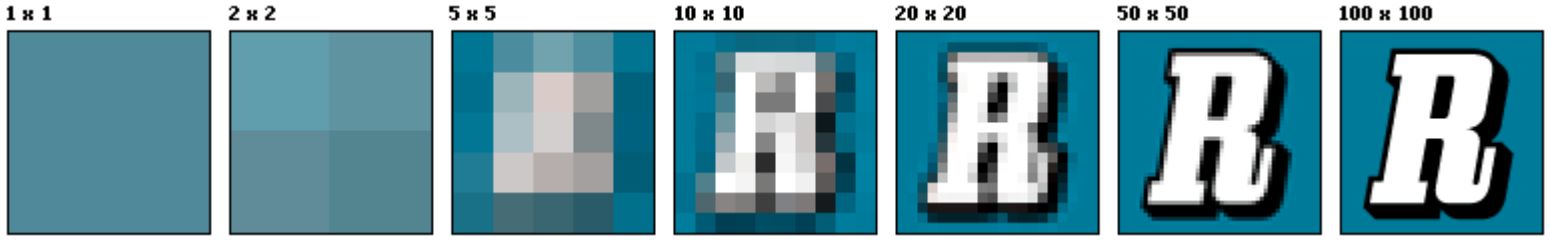
What computers see

How is an image represented digitally?

- Discretise an image by M (columns/width) x N (rows/height) pixels

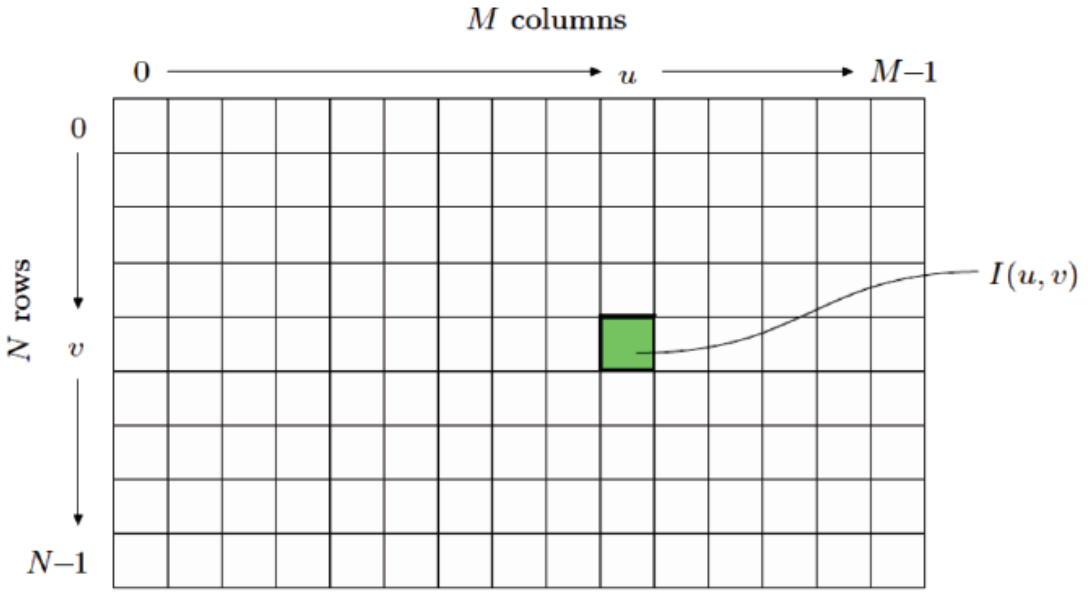
- Pixel resolution: Number of total pixels
  - 1024 x 768
  - 1920 x 1080
  - 3840 x 2160 (4K)





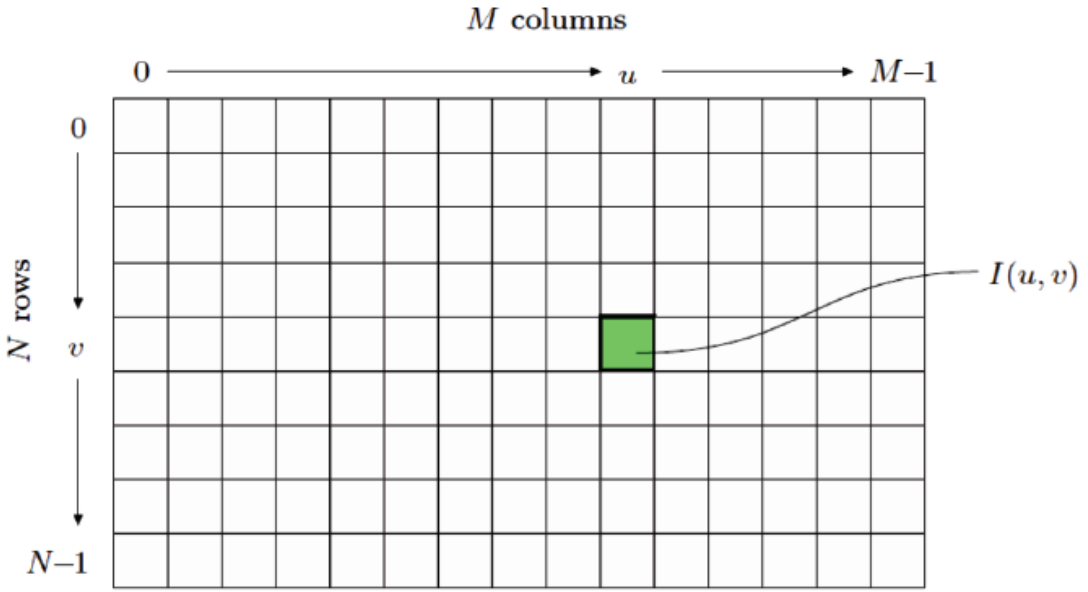
How is an image represented digitally?

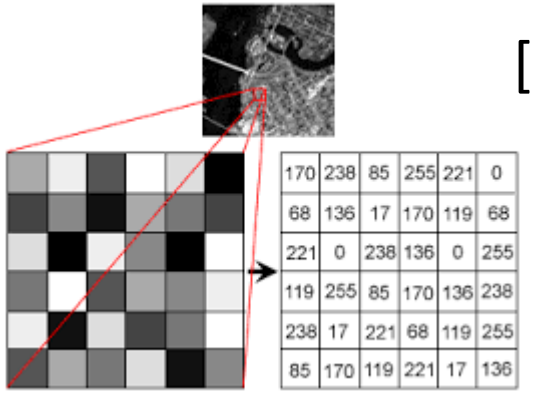
- Discretise an image by M (columns/width) x N (rows/height) pixels
- Pixel resolution: Number of total pixels
- Pixel values: I(u,v) or I(x,y)
  - Sometimes the index order is reversed!
  - Each pixel is associated with an integral number (grayscale image) or an array of integral numbers (colour image)



How is an image represented digitally?

- Discretise an image by M (columns/width) x N (rows/height) pixels
- Pixel resolution: Number of total pixels
- Pixel values: I(u,v) or I(x,y)
  - Sometimes the index order is reversed!
  - Each pixel is associated with an integral number (grayscale image) or an array of integral numbers (colour image)
  - Grayscale image – Intensity of the BLACK colour (0~255)

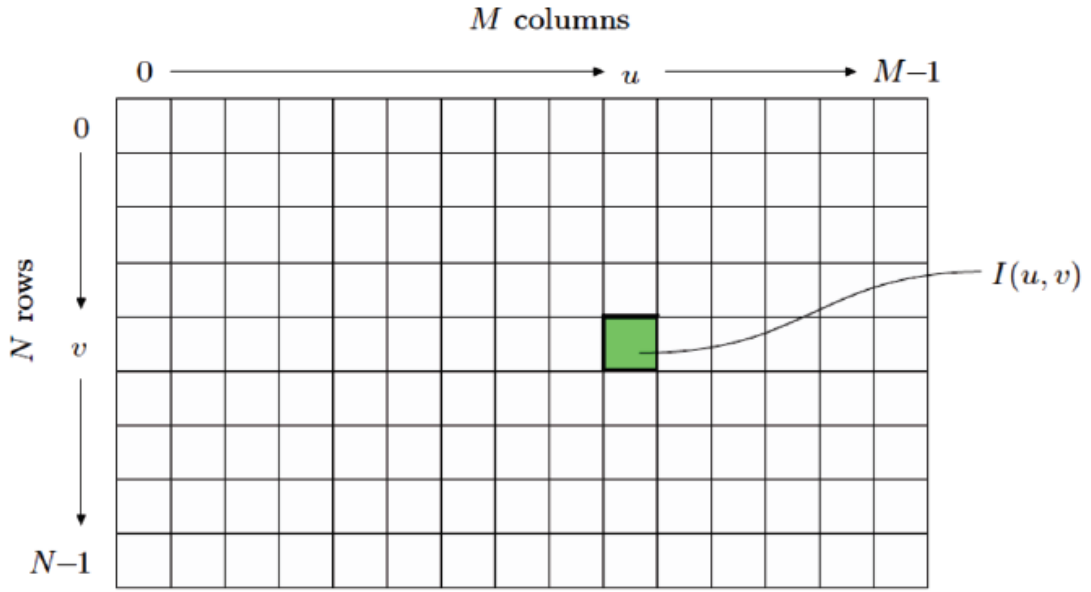


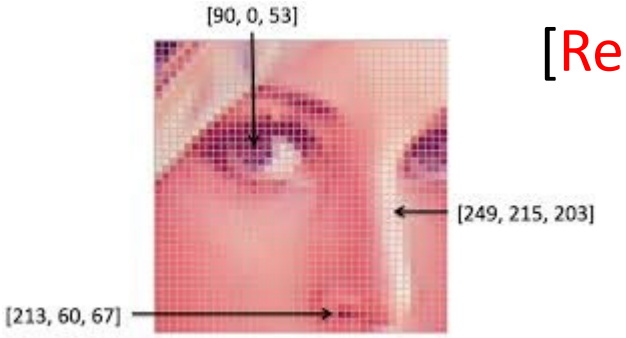


[Black]

How is an image represented digitally?

- Discretise an image by M (columns/width) x N (rows/height) pixels
- Pixel resolution: Number of total pixels
- Pixel values: I(u,v) or I(x,y)
  - Sometimes the index order is reversed!
  - Each pixel is associated with an integral number (grayscale image) or an array of integral numbers (colour image)
  - Colour image – Intensity of the RED, GREEN and BLUE channels (0~255)





[Red, Green, Blue]

OpenCV examples

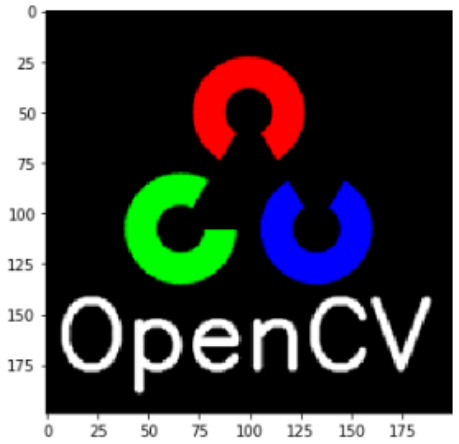
Import libraries

```
import cv2 # OpenCV library
import numpy as np # Numpy library for scientific computing
import matplotlib.pyplot as plt # Matplotlib library for plotting
# show plots in notebook
%matplotlib inline
```

Accessing pixel values

```
img = cv2.imread('opencv_logo.png')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # OpenCV reads an image in the BGR order by default, this function can change the order to RGB
plt.figure(figsize = (9, 5))
plt.imshow(img_rgb)
```

```
<matplotlib.image.AxesImage at 0x7f55321ab0f0>
```



Note the order of the indices!

```
px_rgb = img_rgb[30, 100] # pixel value at row = 30, column = 100
print(px_rgb)
```

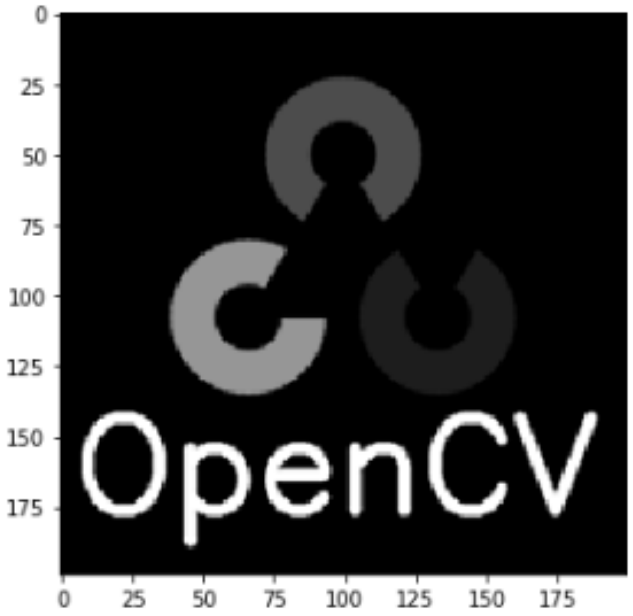
```
[255   0   0]
```

OpenCV examples

Grayscale image

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Change the color image to a grayscale image
plt.figure(figsize = (9, 5))
plt.imshow(img_gray, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f55321121d0>
```



Only one value for each pixel in a grayscale image

```
px_gray = img_gray[30, 100] # pixel value at row = 30, column = 100
print(px_gray)
```

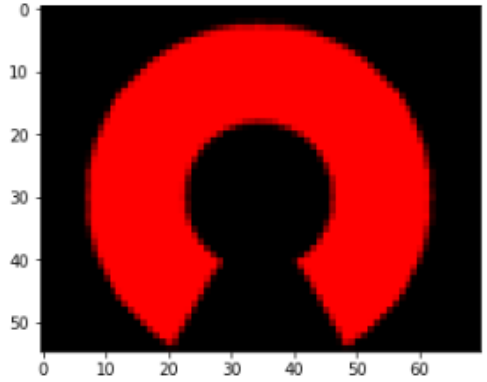
```
76
```

OpenCV examples

Image ROI

```
ROI = img_rgb[20:75, 65:135] # region of interest, rows: 20 - 75, columns: 65 - 135
plt.imshow(ROI)
```

```
<matplotlib.image.AxesImage at 0x7f55320e2d30>
```

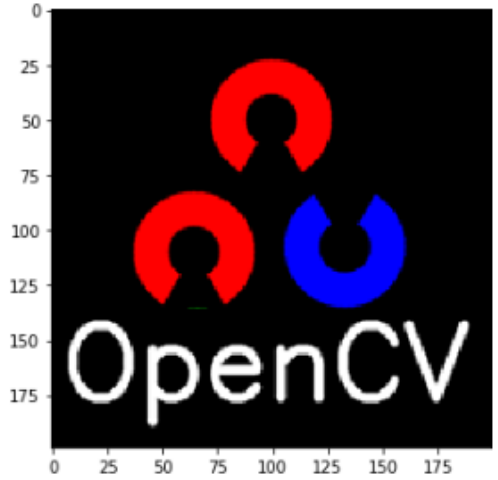


Create a Region of Interest (ROI)

Modify pixel values

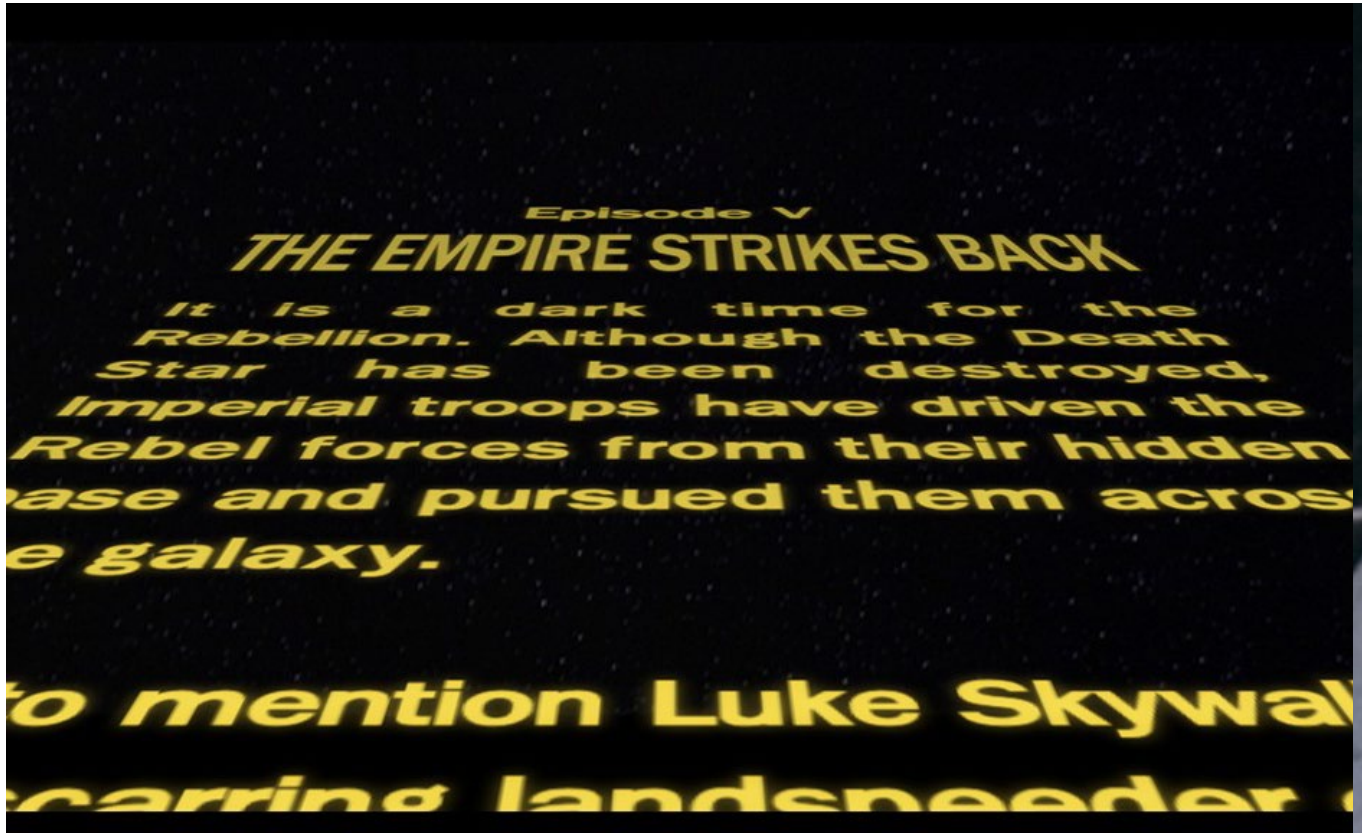
```
img_rgb[80:135, 30:100] = ROI # replace the area of rows: 80 - 135, columns: 30 - 100 with the ROI
plt.figure(figsize = (9, 5))
plt.imshow(img_rgb)
```

```
<matplotlib.image.AxesImage at 0x7f5531dc40b8>
```



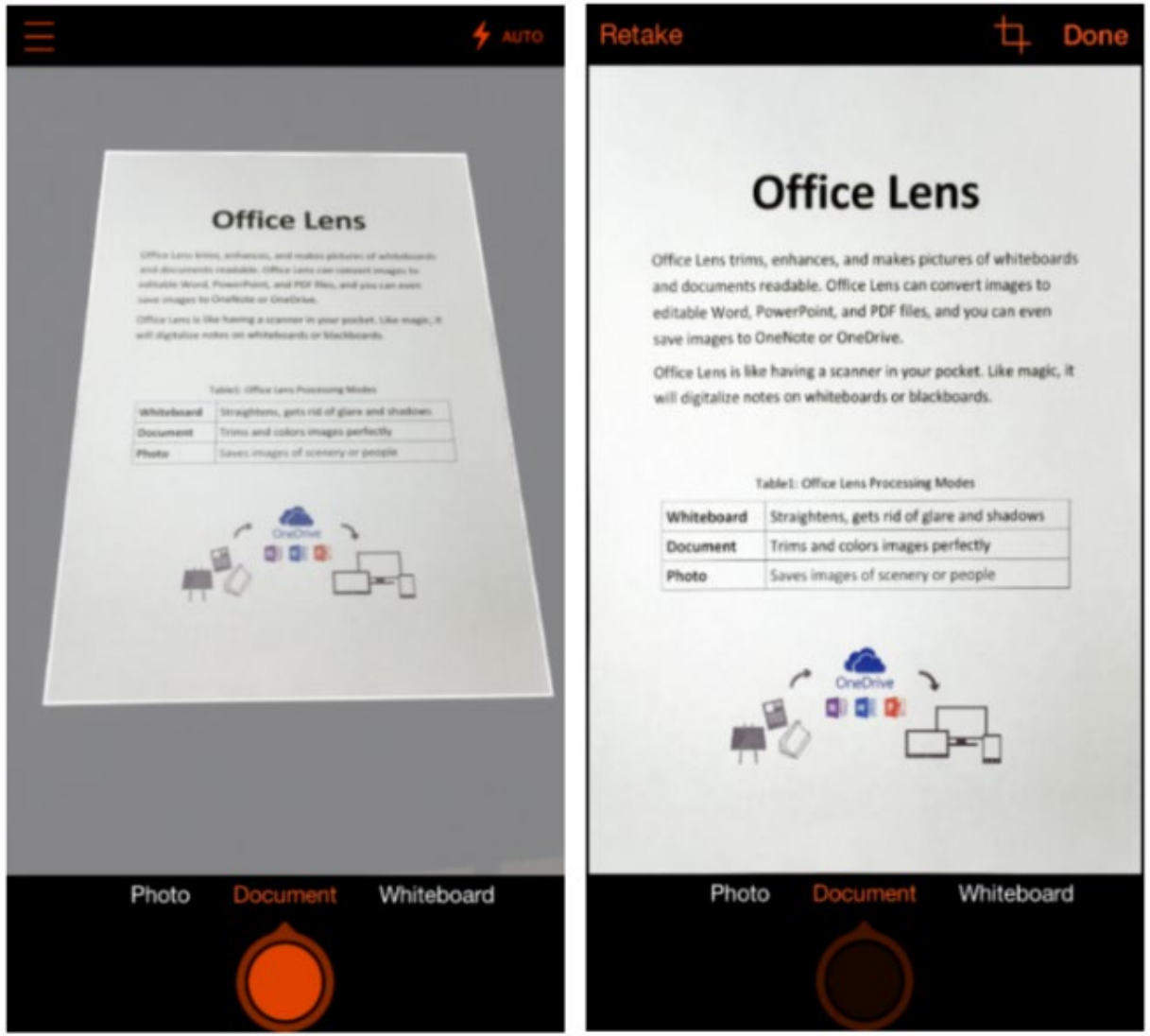
Perspective Transform

Perspective: The angle between the camera and the object



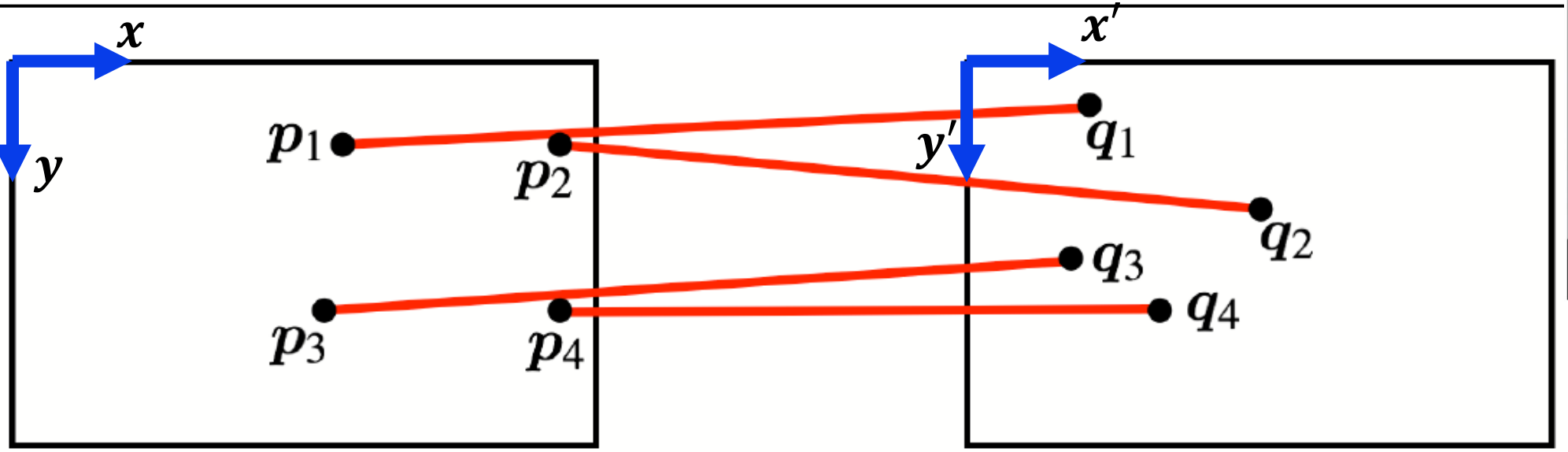
https://www.rocketstock.com/blog/after-effects-video-tutorial-create-star-wars-rolling-titles/

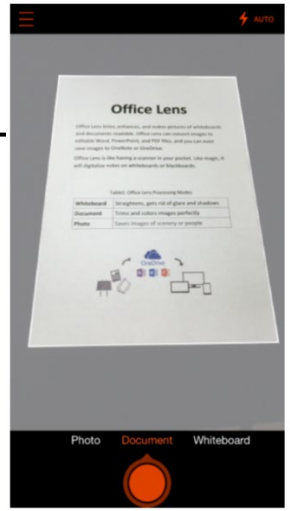
Example application: Office lens



https://www.microsoft.com/en-us/microsoft-365/blog/wp-content/uploads/2015/04/Office-Lens-1-v2.png

Perspective transform





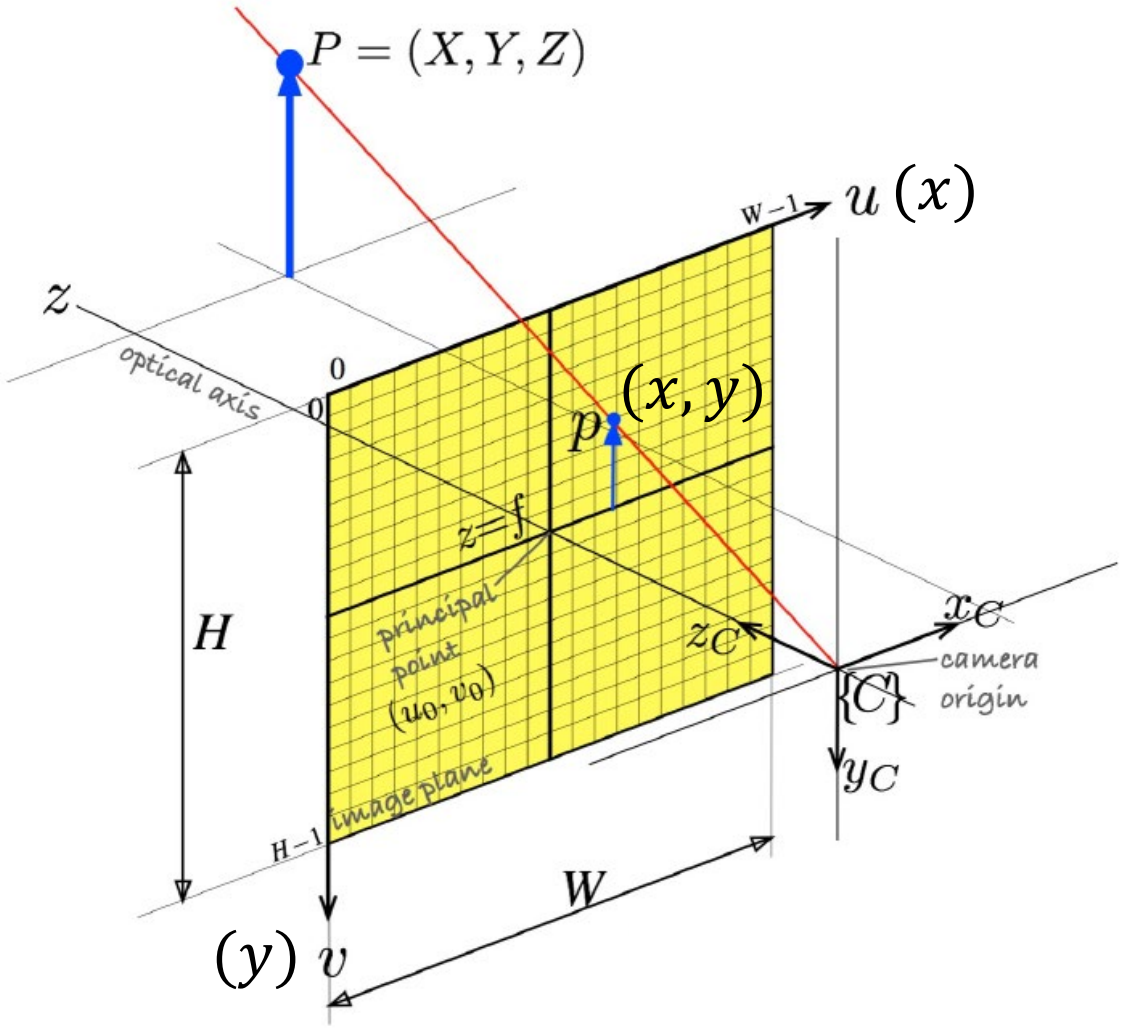
$p_i = (x_i \quad y_i)$

$q_i = (x'_i \quad y'_i)$

Suppose $p_1 – p_4$ are four points on the same plane of the left image, how do we transform the left image into the right image so that $p_1 – p_4$ are changed to $q_1 – q_4$?

https://www.microsoft.com/en-us/microsoft-365/blog/wp-content/uploads/2015/04/Office-Lens-1-v2.png

Pinhole camera projection model



$$x = \frac{fX}{Z} \qquad y = \frac{fY}{Z}$$

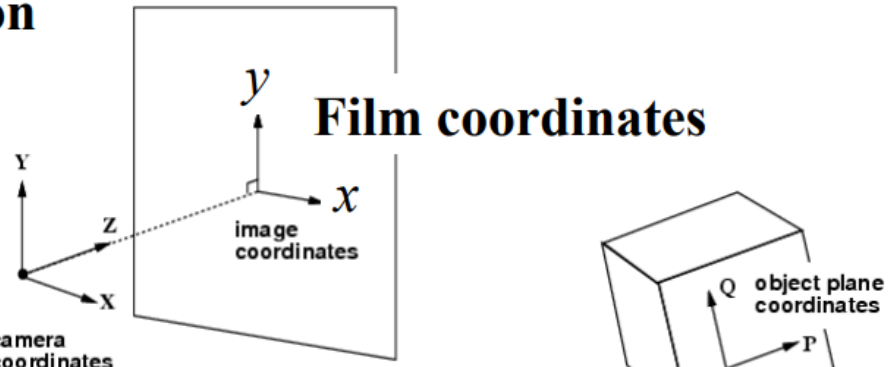
$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{Z} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = g \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Add rigid-body transform between object frame and camera frame

Perspective projection

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$$

Point on plane

Rotation + Translation

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$$

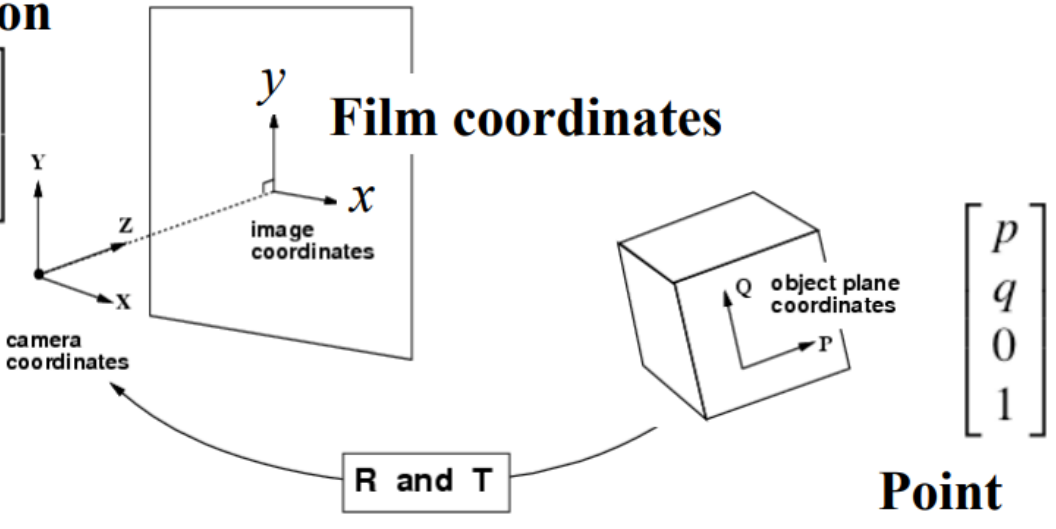
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

Add rigid-body transform between object frame and camera frame

Perspective projection

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{bmatrix} p \\ q \\ 0 \\ 1 \end{bmatrix}$$

Point on plane

Rotation + Translation

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

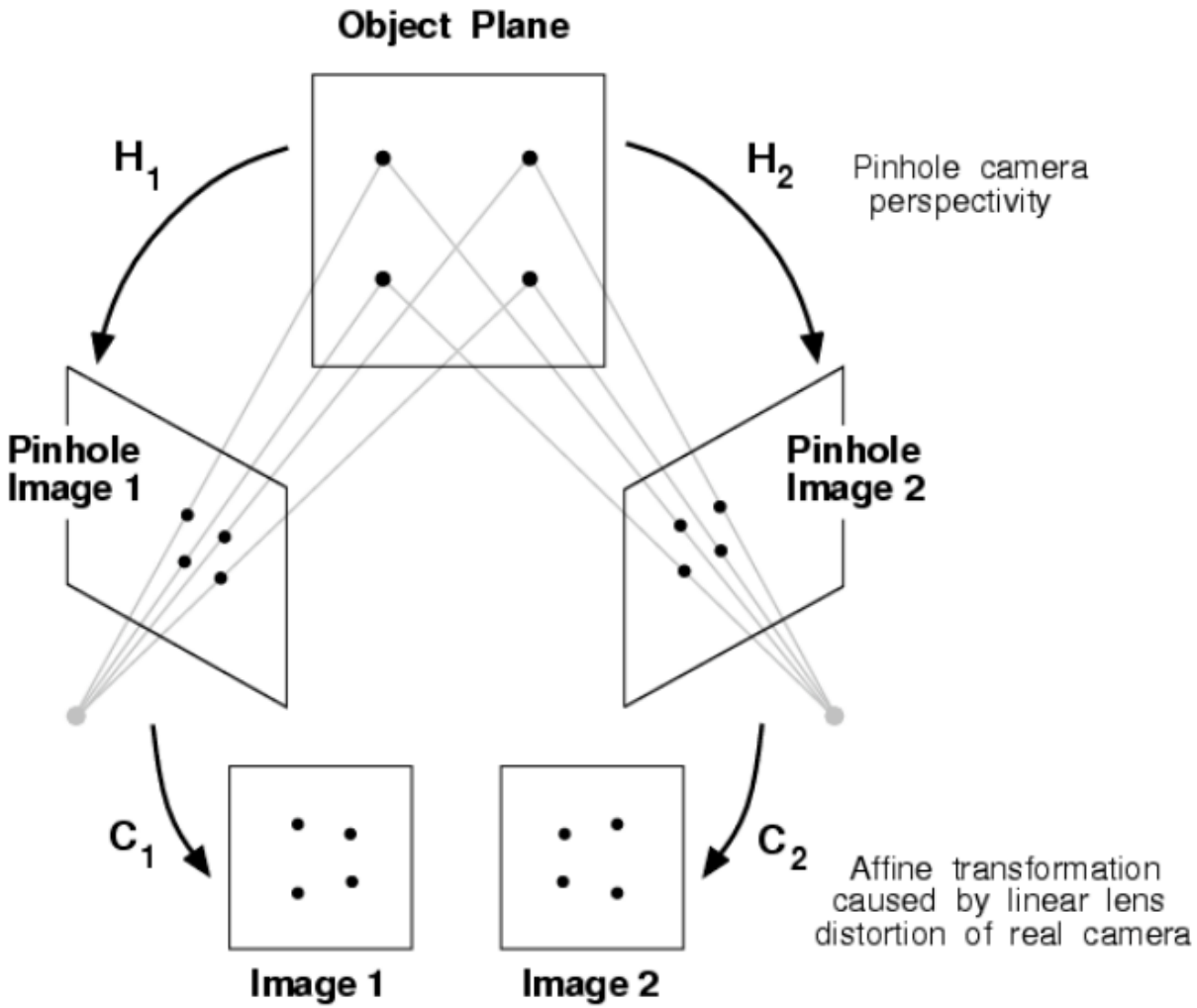
$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} fr_{11} & fr_{12} & ft_x \\ fr_{21} & fr_{22} & ft_y \\ r_{31} & r_{32} & t_z \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

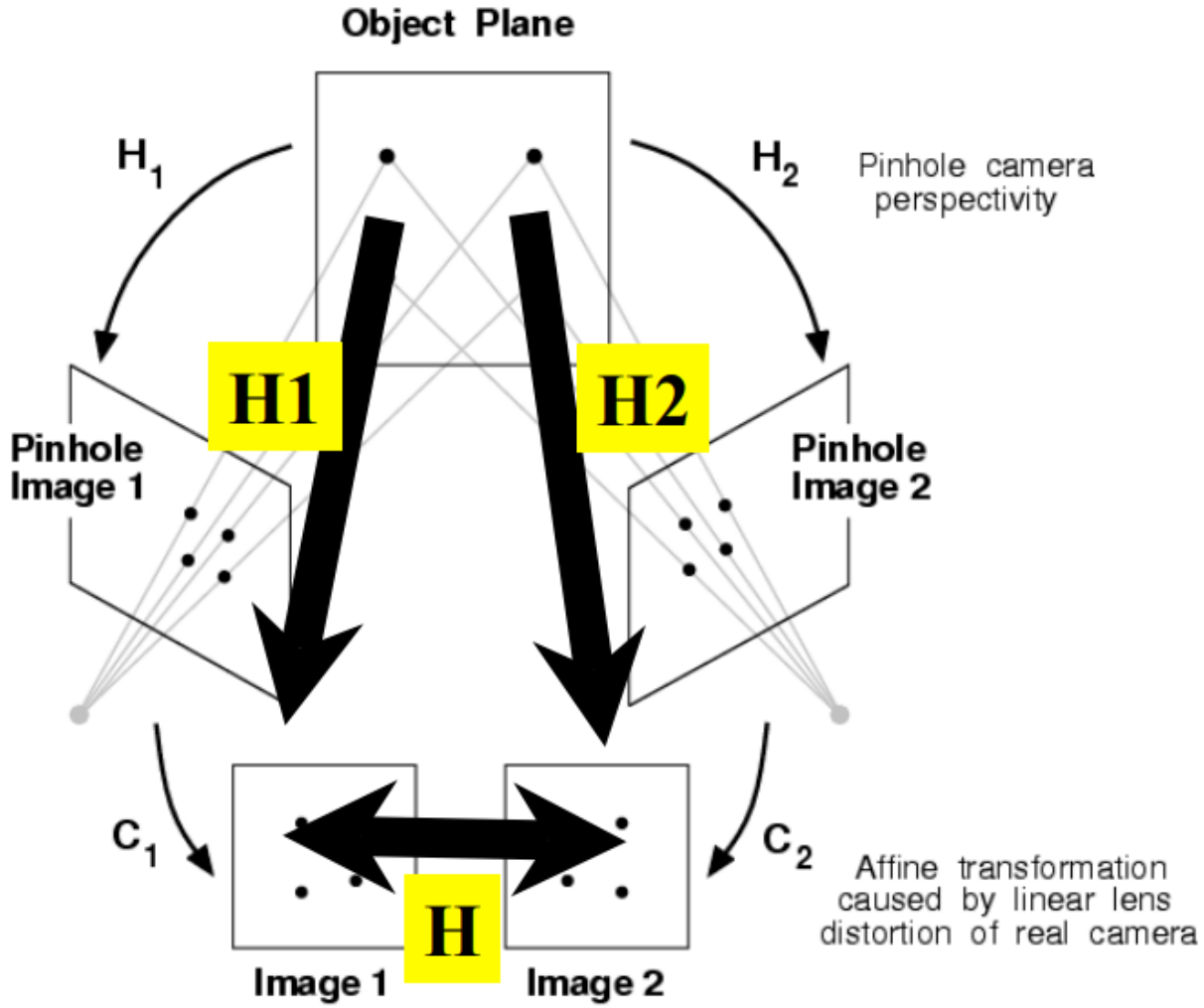
Homography H (planar projective transformation)

Here g is a "point-dependant (depending on the values of p and q on the right side)" scaler to make the last row of the equation hold.

Planar Homography



Planar Homography



$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = g \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} p \\ q \\ 1 \end{bmatrix}$$

- Homography matrices are invertible.
- Product of Homography matrices is also a Homography matrix.

$$H = H_1^{-1} H_2$$

Estimating Homography matrix

Matrix Form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = g \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Coordinates of a point on one plane

Coordinates of a point on another plane

Equations:

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \qquad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \qquad g = \frac{1}{h_{31}x + h_{32}y + h_{33}}$$

Estimating Homography matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = g \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Coordinates of a point on one plane

Coordinates of a point on another plane

There are 9 numbers $h_{11},\ldots,h_{33}$, so are there 9 DOF?

No. Note that we can multiply all h_{ij} by nonzero k without changing the equations:

$$x' = \frac{kh_{11}x + kh_{12}y + kh_{13}}{kh_{31}x + kh_{32}y + kh_{33}} \qquad y' = \frac{kh_{21}x + kh_{22}y + kh_{23}}{kh_{31}x + kh_{32}y + kh_{33}}$$

$$\Rightarrow \quad x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \qquad y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

slido

How many point pairs are needed at least to estimate the Homography matrix?

ⓘ Start presenting to display the poll results on this slide.

Estimating Homography matrix

Setting $h_{33} = 1$

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + 1}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + 1}$$

Multiplying through by denominator

$$(h_{31}x + h_{32}y + 1)x' = h_{11}x + h_{12}y + h_{13}$$

$$(h_{31}x + h_{32}y + 1)y' = h_{21}x + h_{22}y + h_{23}$$

Rearrange

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' = x'$$

$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' = y'$$

Estimating Homography matrix

$$\begin{array}{l} \\ \textbf{Point 1} \\ \\ \textbf{Point 2} \\ \\ \textbf{Point 3} \\ \\ \textbf{Point 4} \end{array} \overset{\textbf{2N x 8}}{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}} \overset{\textbf{8 x 1}}{\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}} = \overset{\textbf{2N x 1}}{\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}}$$

additional points

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

http://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf

Estimating Homography matrix

Linear equations

$$\underset{2N\times 8}{A}\ \underset{8\times 1}{h} = \underset{2N\times 1}{b}$$

Solve:

$$\underset{8\times 2N}{A^T}\ \underset{2N\times 8}{A}\ \underset{8\times 1}{h} = \underset{8\times 2N}{A^T}\ \underset{2N\times 1}{b}$$

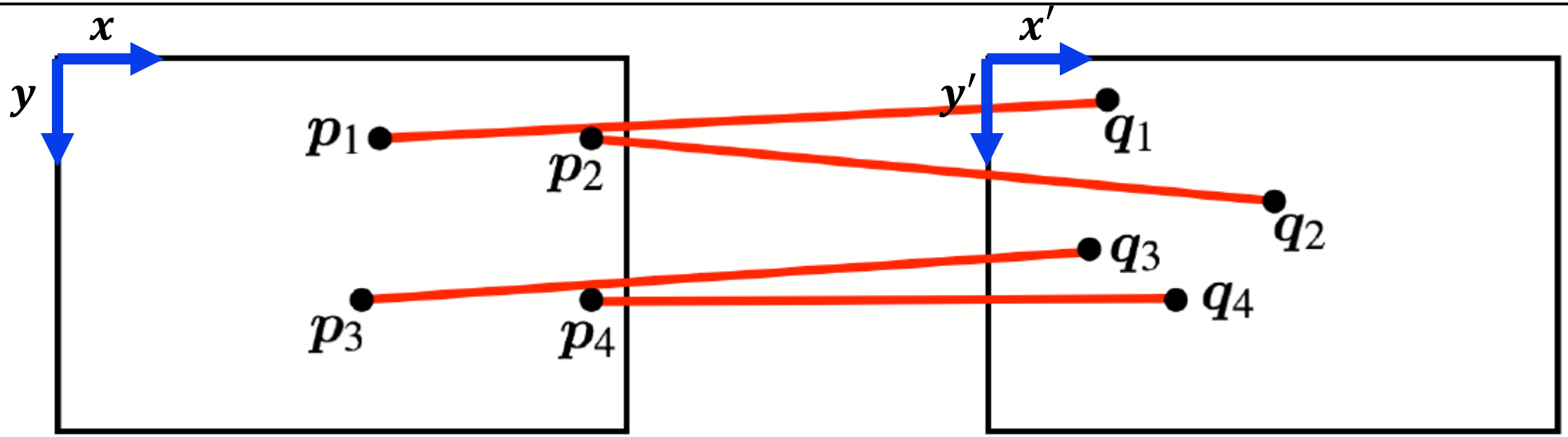
$$\overbrace{(A^T\ A)}^{8\times 8}\ \underset{8\times 1}{h} = \overbrace{(A^T\ b)}^{8\times 1}$$

$$h = (A^T\ A)^{-1}\ (A^T\ b)$$

Matlab: $h = A \backslash b$

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix}$$

With H obtained, the perspective transform can be calculated by:



$$p_i = (x_i \quad y_i) \qquad q_i = (x'_i \quad y'_i)$$

$$\begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix} = g_1 \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x'_2 \\ y'_2 \\ 1 \end{bmatrix} = g_2 \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

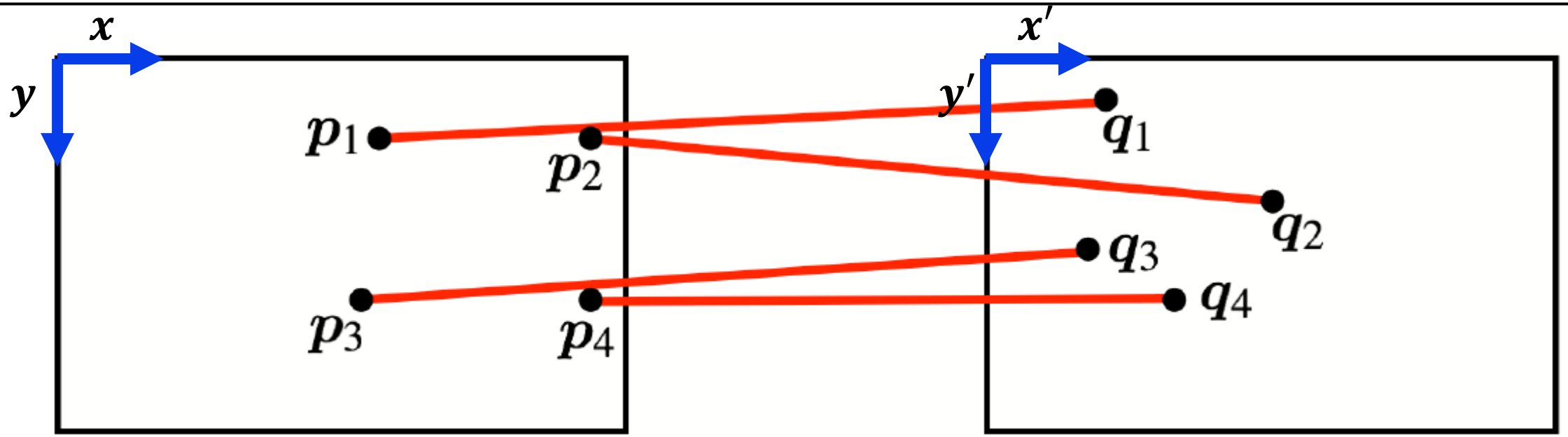
$$\begin{bmatrix} x'_3 \\ y'_3 \\ 1 \end{bmatrix} = g_3 \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x'_4 \\ y'_4 \\ 1 \end{bmatrix} = g_4 \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_4 \\ y_4 \\ 1 \end{bmatrix}$$

where,

$$g_i = \frac{1}{h_{31}x_i + h_{32}y_i + 1}$$

Or in a matrix form:



$$\begin{bmatrix} x'_1 & x'_2 & x'_3 & x'_4 \\ y'_1 & y'_2 & y'_3 & y'_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} g_1 & 0 & 0 & 0 \\ 0 & g_2 & 0 & 0 \\ 0 & 0 & g_3 & 0 \\ 0 & 0 & 0 & g_4 \end{bmatrix}$$

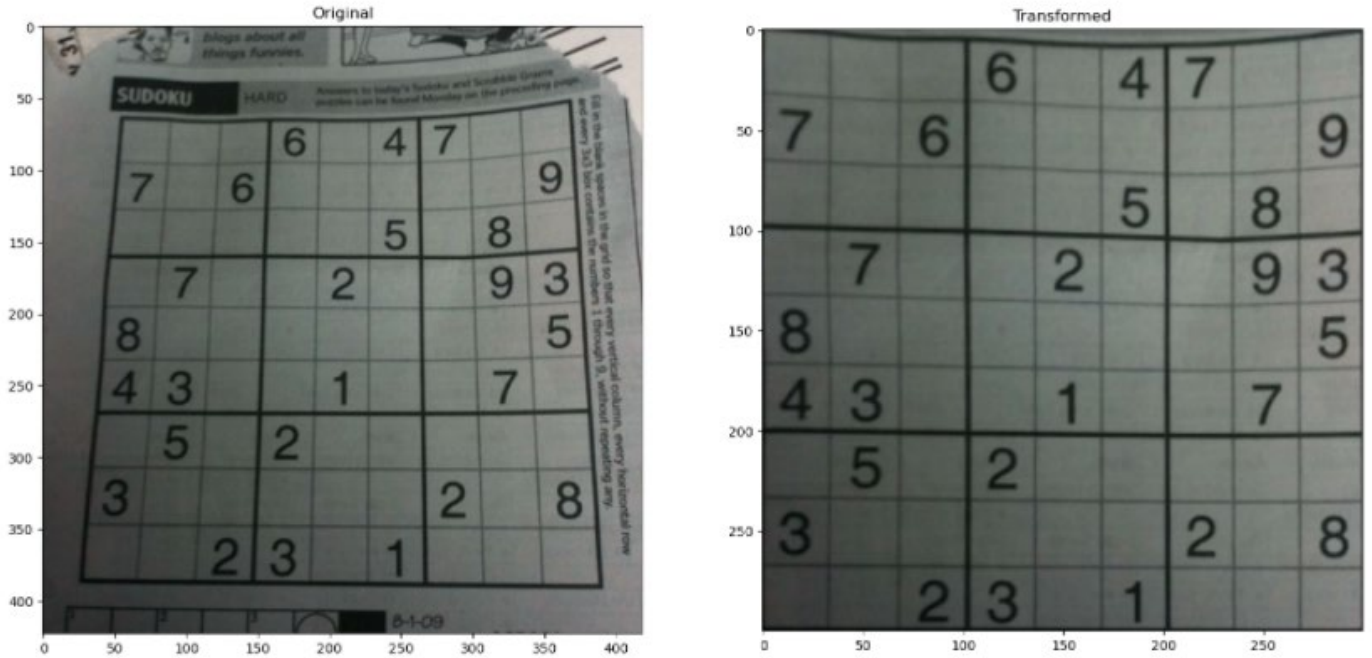
where,

$$g_i = \frac{1}{h_{31}x_i + h_{32}y_i + 1}$$

OpenCV examples

```
## Perspective tranformation
img = cv2.imread('sudoku.jpg') # read an image

pts1 = np.float32([[56,65],[368,52],[28,387],[389,390]]) # four points on the first image
pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]]) # four points on the second image

H = cv2.getPerspectiveTransform(pts1,pts2) # homography matrix
print(H)

dst = cv2.warpPerspective(img, H, (300,300))

fig, (ax1, ax2) = plt.subplots(figsize = (18, 10), ncols = 2)
ax1.imshow(img), ax1.set_title("Original")
ax2.imshow(dst), ax2.set_title("Transformed")
plt.show()
```

```
[[ 1.05587376e+00  9.18151097e-02 -6.50969128e+01]
 [ 4.69010049e-02  1.12562412e+00 -7.57920240e+01]
 [ 1.83251448e-04  5.13337001e-04  1.00000000e+00]]
```



The correspondence must be correct!

Note the coordinates of the points are in the (**u/x/column**, **v/y/row**) order!

Homework:

Write a program in MATLAB (or any other language) to calculate the H matrix given two sets of points.

Colour spaces

Colour

- Three primary colours
  - Red, Green, Blue
- Three secondary colours
  - Cyan, Magenta, Yellow
  - A mixture of two primary colours
- All the other visible colours
  - A mixture of three primary colours

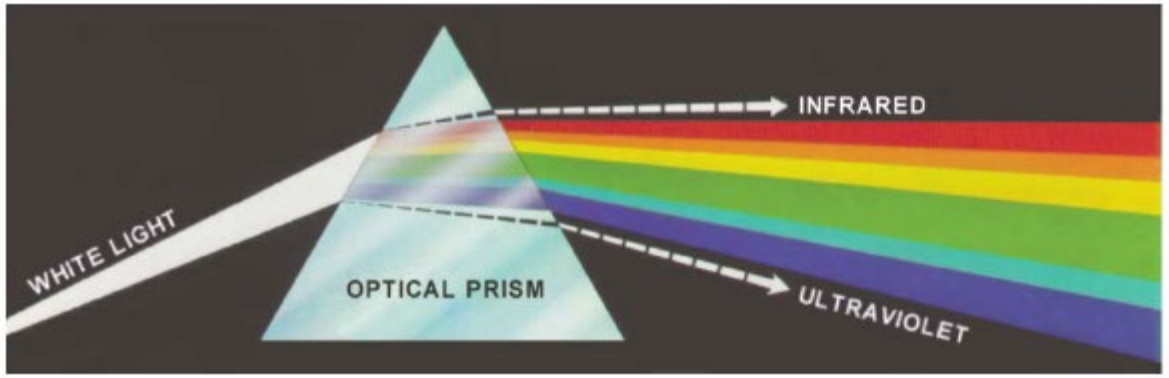
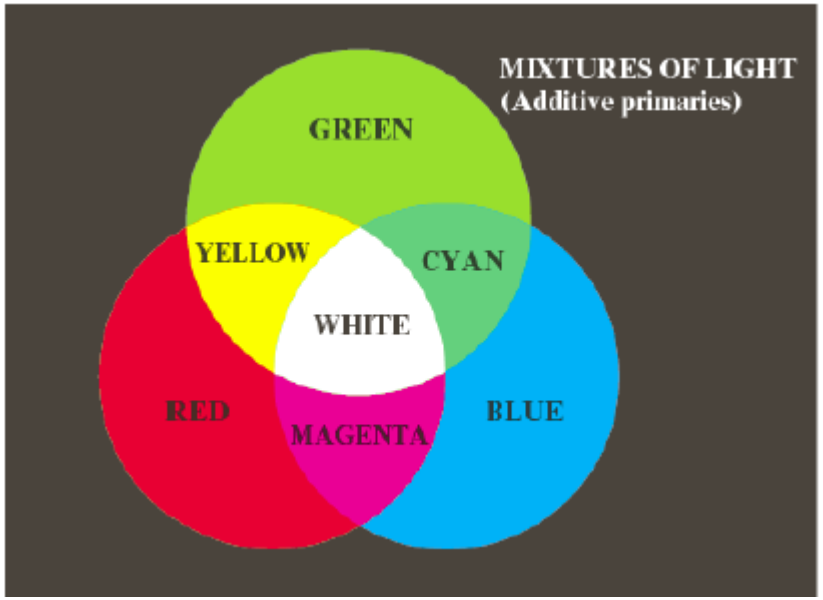


FIGURE 6.1 Color spectrum seen by passing white light through a prism. (Courtesy of the General Electric Co., Lamp Business Division.)



Colour spaces

- Colour space, or colour model/colour system, is a specification of (1) a coordinate system, and (2) a subspace within that system, such that each colour in the space is represented by a single point contained in that subspace.

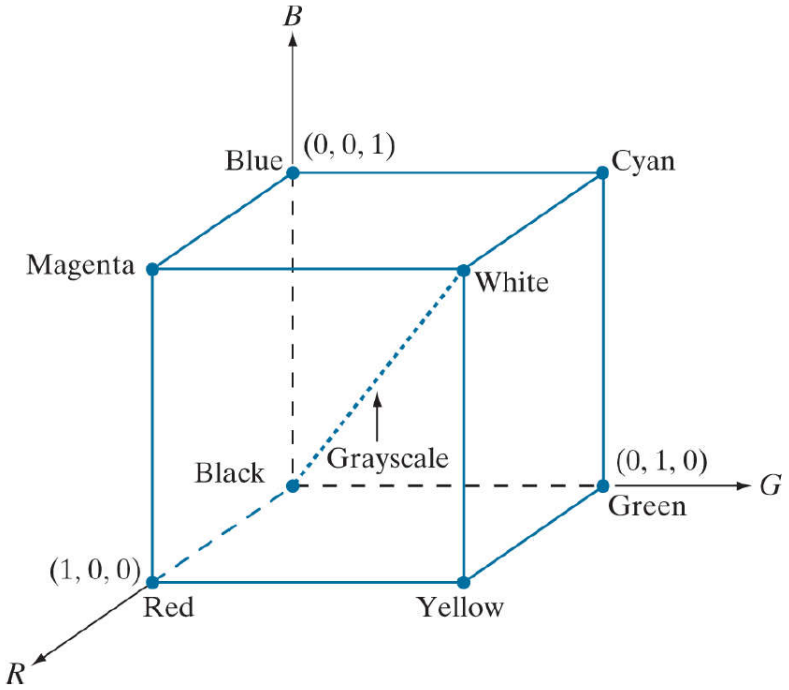


FIGURE 7.7
Schematic of the RGB color cube. Points along the main diagonal have gray values, from black at the origin to white at point (1, 1, 1).

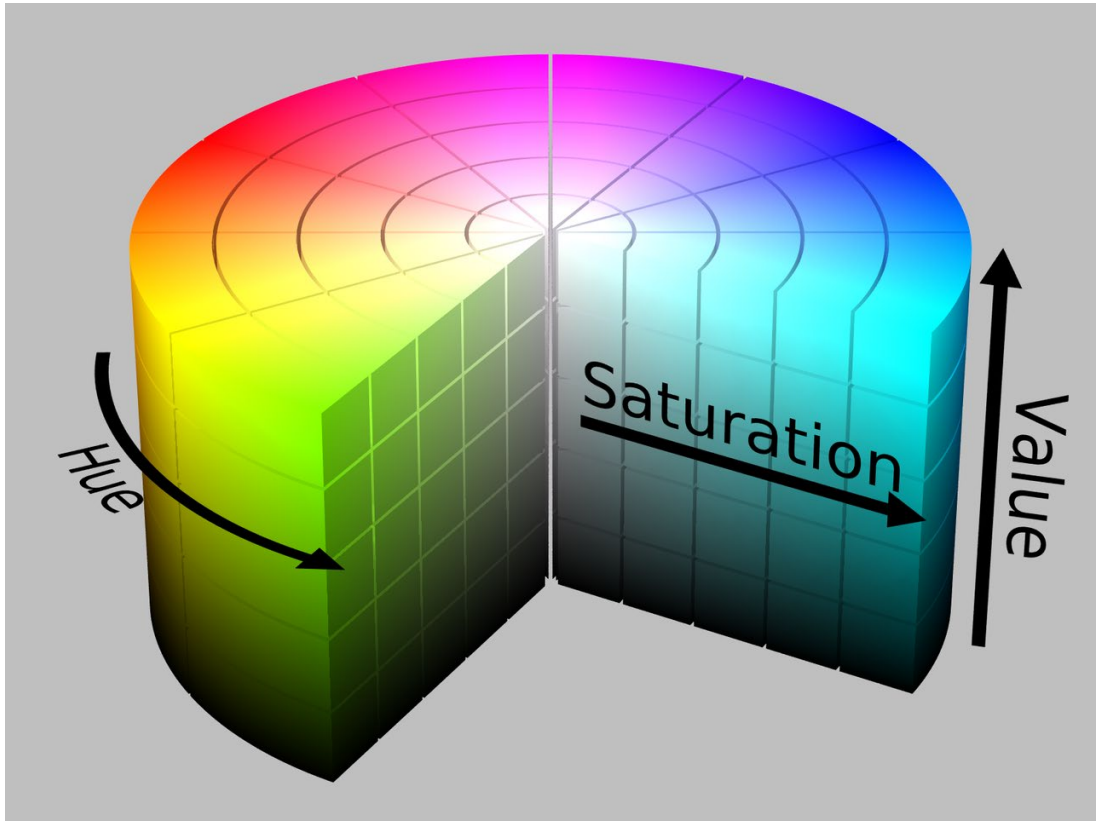
FIGURE 7.8
A 24-bit RGB color cube.

Is RGB always a good colour space to use?

- These are the “same” pink under different illumination.
- In the RGB colour space, all the R, G, B values vary with brightness.

Hue, Saturation, Value (HSV)

- Hue: the type of the colour (the dominant colour observed)
  - 0 – 179 (in OpenCV)
- Saturation: the intensity of the colour (inverse to how much the colour is polluted by the white colour)
  - 0 – 255 (in OpenCV)
- Value: the brightness of the colour
  - 0 – 255 (in OpenCV)



https://en.wikipedia.org/wiki/HSL_and_HSV

Conversion between RGB and HSV

RGB $\leftrightarrow$ HSV

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R,G,B)$$

$$S \leftarrow \begin{cases} \frac{V-min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/(V-min(R,G,B)) & \text{if } V = R \\ 120+60(B-R)/(V-min(R,G,B)) & \text{if } V = G \\ 240+60(R-G)/(V-min(R,G,B)) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

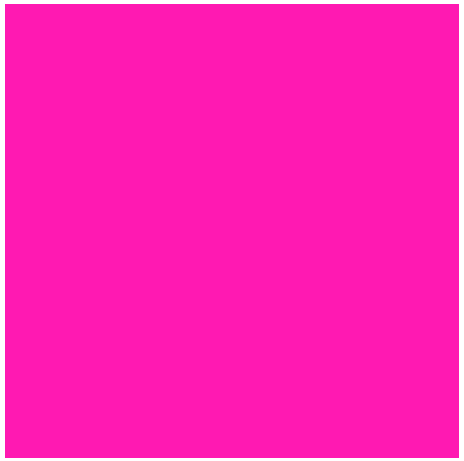
The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255V, S \leftarrow 255S, H \leftarrow H/2 (\text{to fit to 0 to 255})$
- 16-bit images: (currently not supported) $V < -65535V, S < -65535S, H < -H$
- 32-bit images: H, S, and V are left as is

https://docs.opencv.org/4.3.0/de/d25/imgproc_color_conversions.html

Is RGB always a good colour space to use?

Pink?

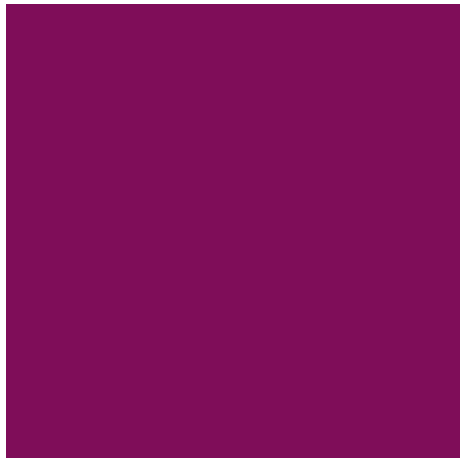


100% brightness
RGB: (255, 25, 178)
HSV: (160, 230, 255)

Same pink?

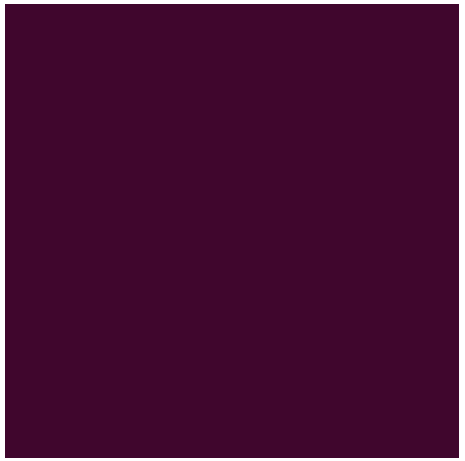
75% brightness
RGB: (191, 19, 134)
HSV: (160, 230, 191)

Same pink again?



50% brightness
RGB: (127, 13, 89)
HSV: (160, 230, 127)

Same pink yet again?



25% brightness
RGB: (64, 6, 45)
HSV: (160, 230, 64)

- These are the "same" pink under different illumination.
- In the RGB colour space, all the R, G, B values vary with brightness.
- In the HSV colour space, only the V value varies with brightness.

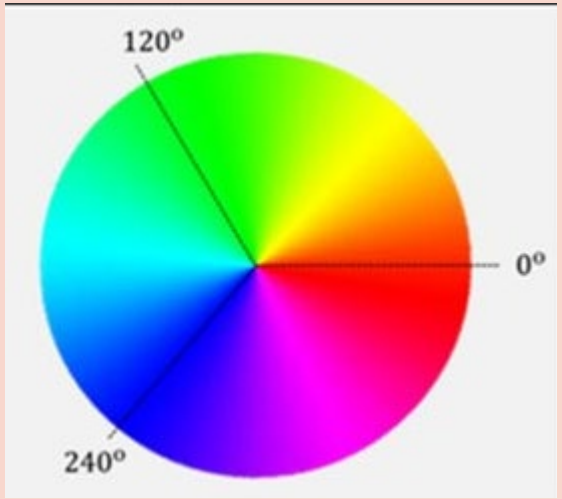
HSV vs RGB

Advantages

- More intuitive (mix colours more like humans do)
  - Imagine you are asked to specify the values of a dark blue
- Separate the chrominance components (H and S) from the brightness (V)
- Make colour-based segmentation/object detection more robust to brightness
- Widely used in computer graphics, computer vision, image processing, etc.

Disadvantages

- Ignoring much of the complexity of colour appearance
- For hue, discontinuity at 0°/360°
  - In OpenCV, H/2 is used, resulting in discontinuity at 0/179



Nick Lawrance, https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/asl-dam/documents/lectures/autonomous_mobile_robots/spring-2019/Planning_I_2019.pdf

Not just RGB or HSV

- There are many other colour spaces
- No general best, only the most suitable to your application

V·T·E	Color space
	List of color spaces · Color models
CAM	CIECAM02 · iCAM
CIE	XYZ (1931) · RGB (1931) · CAM (2002) · YUV (1960) · UVW (1964) · CIELAB (1976) · CIELUV (1976)
RGB	RGB color space · sRGB · rg chromaticity · Adobe · Wide-gamut · ProPhoto · scRGB · DCI-P3 · Rec. 709 · Rec. 2020 · Rec. 2100
YUV	YUV (PAL) · YDbDr (SECAM · PAL-N) · YIQ (NTSC) · YCbCr (Rec. 601 · Rec. 709 · Rec. 2020 · Rec. 2100) · ICtCp (Rec. 2100) · YPbPr · xvYCC · YCoCg
Other	CcMmYK · CMYK · Coloroid · LMS · Hexachrome · HSL, HSV · HCL · Imaginary color · OSA-UCS · PCCS · RG · RYB
Color systems and standards	ACES · ANPA · Colour Index International (CI list of dyes) · DIC · Federal Standard 595 · HKS · ICC profile · ISCC–NBS · Munsell · NCS · Ostwald · Pantone · RAL (list)

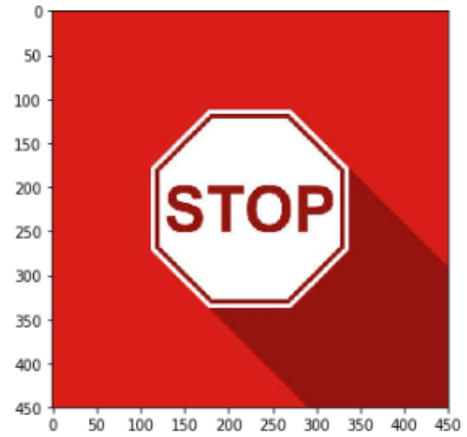
https://en.wikipedia.org/wiki/Color_space

OpenCV examples

Change Colour space to HSV

```
img = cv2.imread('stop_shadow.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # OpenCV reads an image in the BGR order by default, this function can change the order to RGB
plt.figure(figsize = (9, 5))
plt.imshow(img_rgb)
```

```
<matplotlib.image.AxesImage at 0x7f55311b4320>
```




```
px_rgb = img_rgb[100, 100] # pixel value at row = 100, column = 100
print(px_rgb)
px_rgb = img_rgb[400, 400] # pixel value at row = 400, column = 400
print(px_rgb)
```

```
[217  30  25]
[150  21  16]
```

```
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV) # OpenCV reads an image in the BGR order by default, this function can change the order to RGB
px_hsv = img_hsv[100, 100] # pixel value at row = 100, column = 100
print(px_hsv)
px_hsv = img_hsv[400, 400] # pixel value at row = 400, column = 400
print(px_hsv)
```

```
[  1 226 217]
[  1 228 150]
```

RGB to Grayscale

- Instead of handling three channels, sometimes it is easier to process just one channel. Thus we may want to convert a colour image to a grayscale image.

RGB ↔ GRAY

Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

$$\text{RGB[A] to Gray:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

and

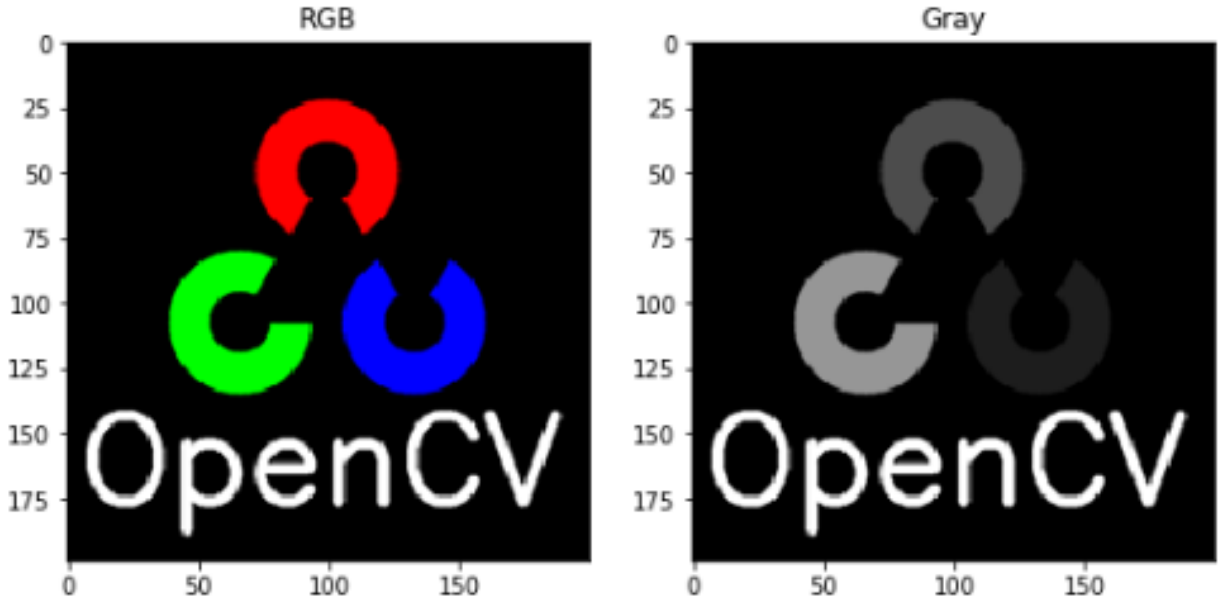
$$\text{Gray to RGB[A]:} \quad R \leftarrow Y, G \leftarrow Y, B \leftarrow Y, A \leftarrow \max(ChannelRange)$$

https://docs.opencv.org/4.3.0/de/d25/imgproc_color_conversions.html

OpenCV examples

```
img = cv2.imread('opencv_logo.png')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # OpenCV reads an image in the BGR order by default, this function can change the order to RGB
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Change the color image to a grayscale image
fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
ax1.imshow(img_rgb), ax1.set_title("RGB")
ax2.imshow(img_gray, cmap='gray'), ax2.set_title("Gray")
```

```
(<matplotlib.image.AxesImage at 0x7f8f0d1047f0>, Text(0.5, 1.0, 'Gray'))
```



RGB[A] to Gray: $Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$

Thresholding

Thresholding

- Modify (keep, remove, change, etc.) pixel values based on some thresholds

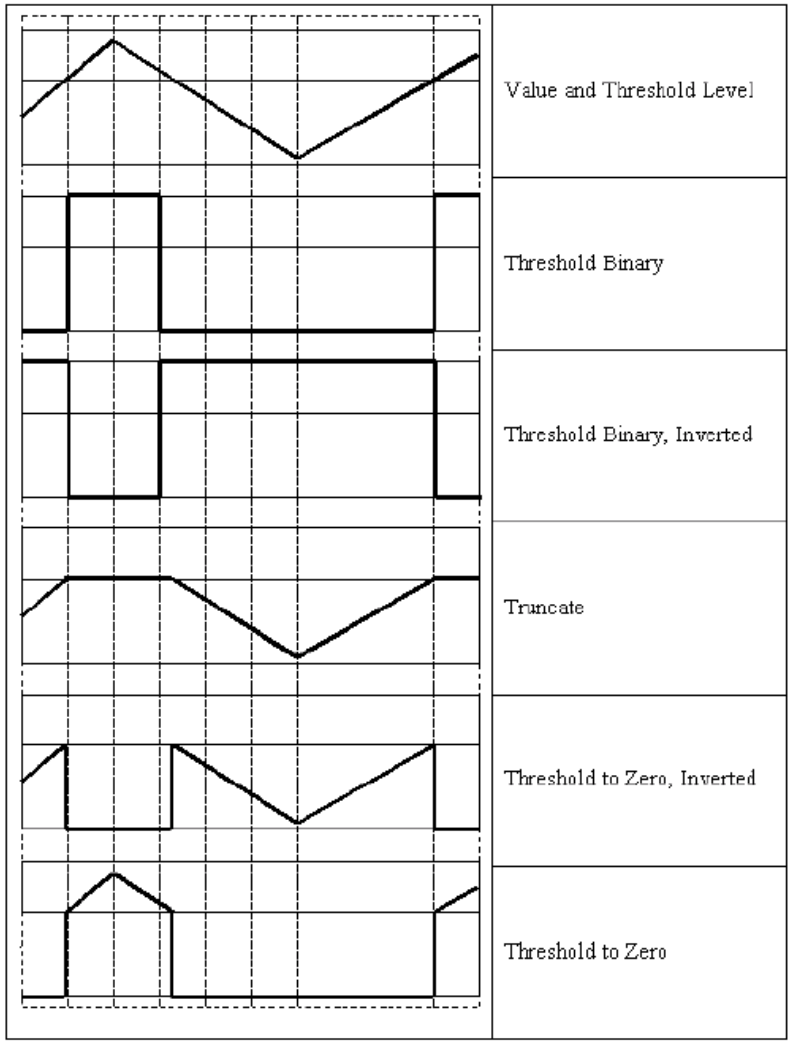
- Grayscale thresholding
  - Thresholding based on the grayscale values

- Colour-based thresholding
  - Thresholding based on the colour values

Grayscale thresholding

retval, dst = cv.threshold(src, thresh, maxval, type[, dst])

Enumerator	
THRESH_BINARY Python: cv.THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV Python: cv.THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC Python: cv.THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO Python: cv.THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV Python: cv.THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_MASK Python: cv.THRESH_MASK	
THRESH_OTSU Python: cv.THRESH_OTSU	flag, use Otsu algorithm to choose the optimal threshold value
THRESH_TRIANGLE Python: cv.THRESH_TRIANGLE	flag, use Triangle algorithm to choose the optimal threshold value

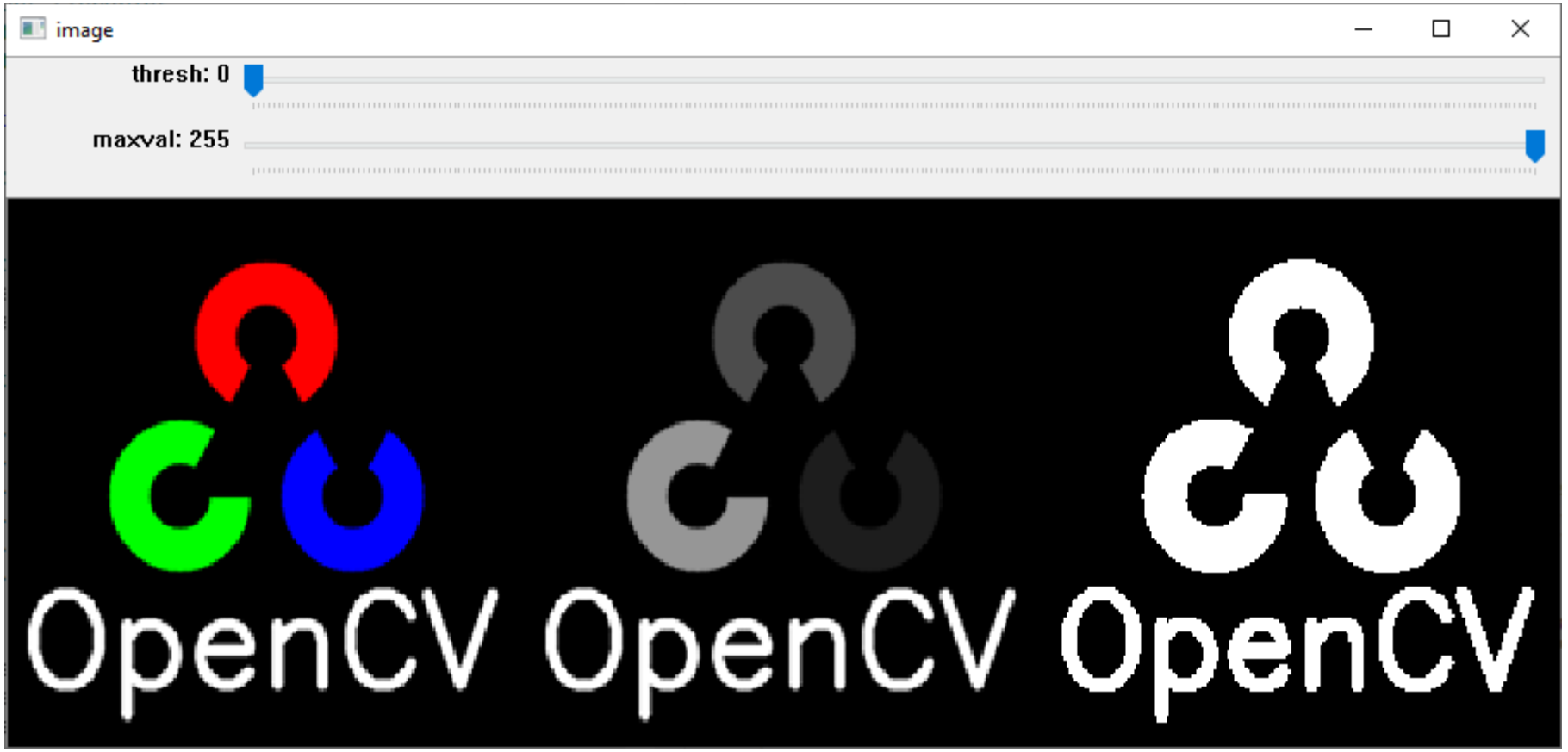


https://docs.opencv.org/master/db/d8e/tutorial_threshold.html

OpenCV examples

```python
# Import libraries
import cv2
import numpy as np

def nothing(x): pass

# Load image
image = cv2.imread('opencv_logo.png')

# Create a window
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.resizeWindow('image', 900, 400)

# Create trackbars for thresholding
# Set thresh and maxval range from 0-255
cv2.createTrackbar('thresh', 'image', 0, 255, nothing)
cv2.createTrackbar('maxval', 'image', 0, 255, nothing)

# Set default value for thresh and maxval trackbars
cv2.setTrackbarPos('thresh', 'image', 0)
cv2.setTrackbarPos('maxval', 'image', 255)

# Initialize thresh and maxval values
thresh = maxval = 0
pthresh = pmaxval = 0

while(1):

    # Get current positions of all trackbars
    thresh = cv2.getTrackbarPos('thresh', 'image')
    maxval = cv2.getTrackbarPos('maxval', 'image')

    # Convert to gray format and threshold
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, threshed = cv2.threshold(gray, thresh, maxval, cv2.THRESH_BINARY)

    # Convert grayscale image to color image for displaying simultaneous
    gray_3_channel = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
    threshed_3_channel = cv2.cvtColor(threshed, cv2.COLOR_GRAY2BGR)

    # Stack images
    numpy_horizontal = np.hstack((image, gray_3_channel, threshed_3_channel))

    # Print if there is a change in threshold or maxval value
    if((pthresh != thresh) | (pmaxval != maxval)):
        print("(thresh = %d , maxval = %d)" % (thresh , maxval))
        pthresh = thresh
        pmaxval = maxval

    # Display stacked image, press q to quit
    cv2.imshow('image', numpy_horizontal)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
```

OpenCV examples



Colour-based thresholding

`dst = cv.inRange( src, lowerb, upperb[, dst] )`

Checks if array elements lie between the elements of two other arrays.

The function checks the range as follows:

- For every element of a single-channel input array:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0$$

- For two-channel arrays:

$$\texttt{dst}(I) = \texttt{lowerb}(I)_0 \leq \texttt{src}(I)_0 \leq \texttt{upperb}(I)_0 \land \texttt{lowerb}(I)_1 \leq \texttt{src}(I)_1 \leq \texttt{upperb}(I)_1$$

- and so forth.

That is, dst (I) is set to 255 (all 1 -bits) if src (I) is within the specified 1D, 2D, 3D, ... box and 0 otherwise.

When the lower and/or upper boundary parameters are scalars, the indexes (I) at lowerb and upperb in the above formulas should be omitted.

Parameters

src	first input array.
lowerb	inclusive lower boundary array or a scalar.
upperb	inclusive upper boundary array or a scalar.
dst	output array of the same size as src and CV_8U type.

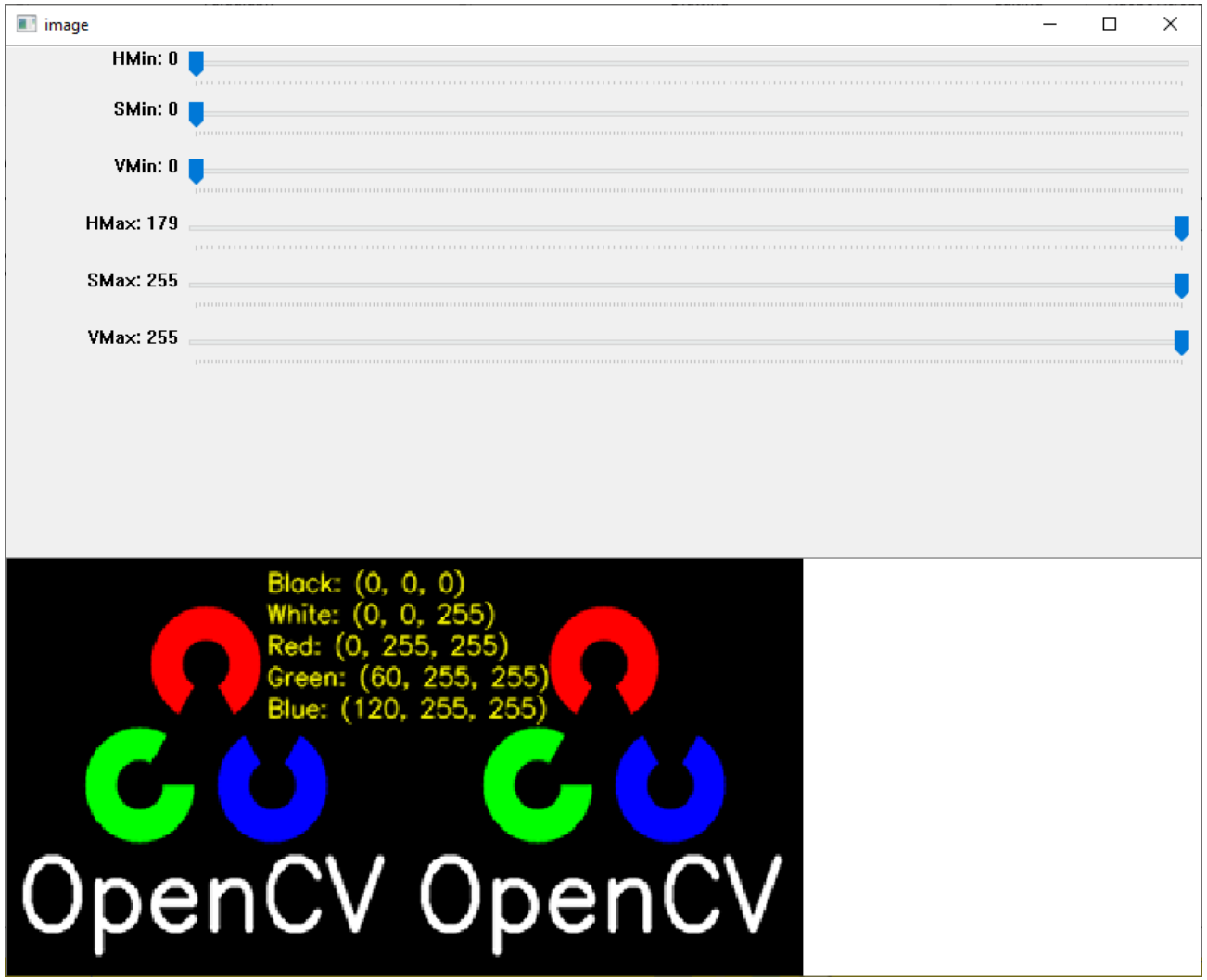
https://docs.opencv.org/master/da/d97/tutorial_threshold_inRange.html

OpenCV examples

```python
# Import libraries
import cv2
import numpy as np

def nothing(x): pass

# Load image
image = cv2.imread('opencv_logo.png')

# Create a window
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.resizeWindow('image', 900, 700)

# Create trackbars for color change
# Hue is from 0-179 for Opencv
cv2.createTrackbar('HMin', 'image', 0, 179, nothing)
cv2.createTrackbar('SMin', 'image', 0, 255, nothing)
cv2.createTrackbar('VMin', 'image', 0, 255, nothing)
cv2.createTrackbar('HMax', 'image', 0, 179, nothing)
cv2.createTrackbar('SMax', 'image', 0, 255, nothing)
cv2.createTrackbar('VMax', 'image', 0, 255, nothing)

# Set default value for Max HSV trackbars
cv2.setTrackbarPos('HMax', 'image', 179)
cv2.setTrackbarPos('SMax', 'image', 255)
cv2.setTrackbarPos('VMax', 'image', 255)

# Initialize HSV min/max values
hMin = sMin = vMin = hMax = sMax = vMax = 0
phMin = psMin = pvMin = phMax = psMax = pvMax = 0

```

```python
while(1):

    # Get current positions of all trackbars
    hMin = cv2.getTrackbarPos('HMin', 'image')
    sMin = cv2.getTrackbarPos('SMin', 'image')
    vMin = cv2.getTrackbarPos('VMin', 'image')
    hMax = cv2.getTrackbarPos('HMax', 'image')
    sMax = cv2.getTrackbarPos('SMax', 'image')
    vMax = cv2.getTrackbarPos('VMax', 'image')

    # Set minimum and maximum HSV values to display
    lower = np.array([hMin, sMin, vMin])
    upper = np.array([hMax, sMax, vMax])

    # Convert to HSV format and color threshold
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    result = cv2.bitwise_and(image, image, mask=mask)

    # Convert grayscale image to color image for displaying simultaneous
    mask_3_channel = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)

    # Stack images
    numpy_horizontal = np.hstack((image, result, mask_3_channel))

    # Display HSV values of some colours
    cv2.putText(numpy_horizontal, 'Black: (0, 0, 0)', (130, 15), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,255,255), 1, cv2.LINE_AA)
    cv2.putText(numpy_horizontal, 'White: (0, 0, 255)', (130, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,255,255), 1, cv2.LINE_AA)
    cv2.putText(numpy_horizontal, 'Red: (0, 255, 255)', (130, 45), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,255,255), 1, cv2.LINE_AA)
    cv2.putText(numpy_horizontal, 'Green: (60, 255, 255)', (130, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,255,255), 1, cv2.LINE_AA)
    cv2.putText(numpy_horizontal, 'Blue: (120, 255, 255)', (130, 75), cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0,255,255), 1, cv2.LINE_AA)

    # Print if there is a change in HSV value
    if((phMin != hMin) | (psMin != sMin) | (pvMin != vMin) | (phMax != hMax) | (psMax != sMax) | (pvMax != vMax) ):
        print("(hMin = %d , sMin = %d, vMin = %d), (hMax = %d , sMax = %d, vMax = %d)" % (hMin , sMin , vMin, hMax, sMax , vMax))
        phMin = hMin
        psMin = sMin
        pvMin = vMin
        phMax = hMax
        psMax = sMax
        pvMax = vMax

    # Display result image, press q to quit
    cv2.imshow('image', numpy_horizontal)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break
cv2.destroyAllWindows()
```

OpenCV examples



Morphological Operations

Morphological operations

- Morphology:
  - The study of shapes
- Morphological operations:
  - Image processing based on the shape of features
- Especially suited to the processing of binary images
- We will introduce four basic operations:
  - Erosion
  - Dilation
  - Opening
  - Closing

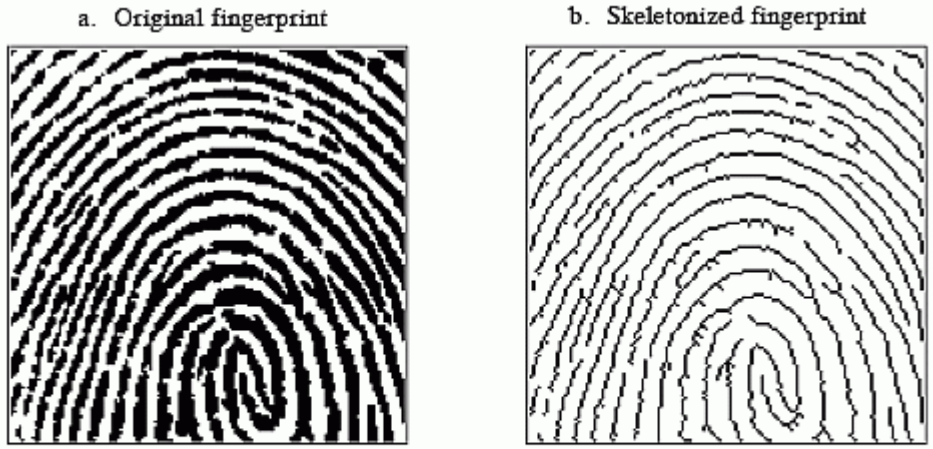


FIGURE 25-11
Binary skeletonization. The binary image of a fingerprint, (a), contains ridges that are many pixels wide. The skeletonized version, (b), contains ridges only a single pixel wide.

Erosion

- Kernel:
  - A small image patch
  - E.g., a 3 x 3 square kernel
- Spatial operations:
  - Slide the kernel through the image
  - Set the central pixel 1 only if all the pixels under the kernel are 1
- Features:
  - Erode edges
  - Thin thickness of objects
  - Remove "salt noise"

1	1	1
1	**1**	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Original

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	**1**	**1**	**1**	**1**	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

After Erosion

Dilation

- Kernel:
  - A small image patch
  - E.g., a 3 x 3 rectangle kernel
- Spatial operations:
  - Slide the kernel through the image
  - Set the central pixel 1 if at least one pixel under the kernel is 1
- Features:
  - Dilate edges
  - Thicken thickness of objects
  - Remove "pepper noise"

1	1	1
1	**1**	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Original

0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	**1**	0	0	0	0
0	**1**	1	1	1	**1**	0	0	0	0
0	**1**	1	1	1	**1**	0	0	0	0
0	**1**	1	1	1	**1**	**1**	**1**	**1**	0
0	**1**	1	1	1	1	1	1	**1**	0
0	**1**	1	**1**	1	1	1	1	**1**	0
0	**1**	1	1	1	1	1	1	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0	0

After Dilation

Opening

- Kernel:
  - A small image patch
  - E.g., a 3 x 3 rectangle kernel
- Spatial operations:
  - First Erosion
  - Then Dilation
- Features:
  - Open weak links
  - Remove "salt noise"

1	1	1
1	**1**	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Original

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	**1**	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	**1**	**1**	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

First Erosion

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	**0**	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	**0**	0	1	1	1	1	0	0
0	0	**0**	**0**	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Then Dilation

Closing

- Kernel:
  - A small image patch
  - E.g., a 3 x 3 rectangle kernel
- Spatial operations:
  - First Dilation
  - Then Erosion
- Features:
  - Close weak links
  - Remove "pepper noise"

1	1	1
1	**1**	1
1	1	1

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Original

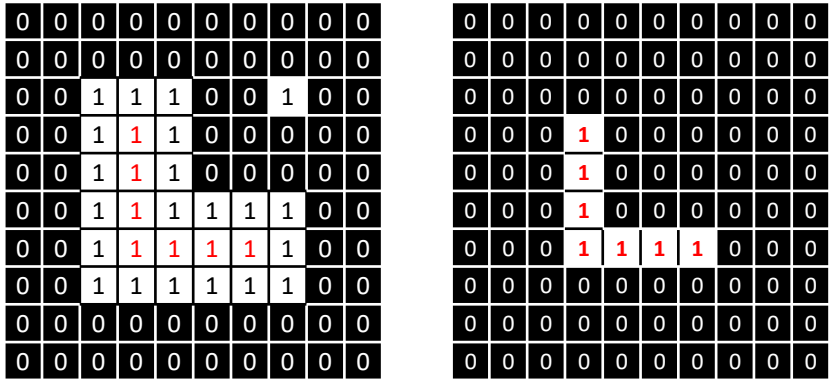
0	0	0	0	0	0	0	0	0	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	**1**	1	1	1	**1**	**1**	1	**1**	0
0	**1**	1	1	1	**1**	**1**	**1**	**1**	0
0	**1**	1	1	1	**1**	**1**	**1**	**1**	0
0	**1**	1	1	1	1	1	1	**1**	0
0	**1**	1	**1**	1	1	1	1	**1**	0
0	**1**	1	1	1	1	1	1	**1**	0
0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0
0	0	0	0	0	0	0	0	0	0

First Dilation

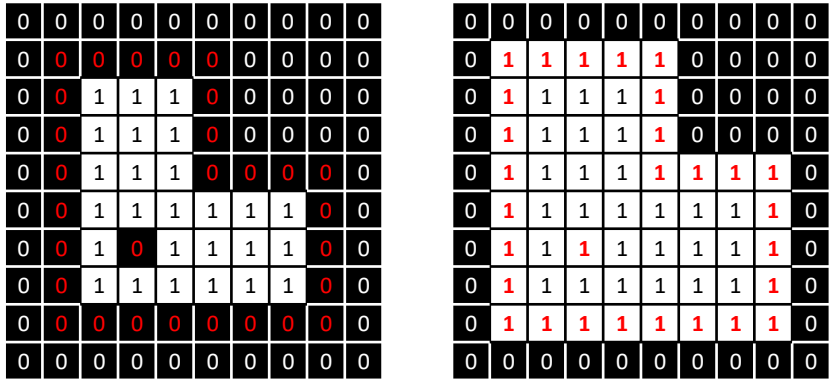
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	**1**	**1**	**1**	0	0
0	0	1	1	1	**1**	**1**	**1**	0	0
0	0	1	1	1	**1**	**1**	**1**	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	**1**	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Then Erosion

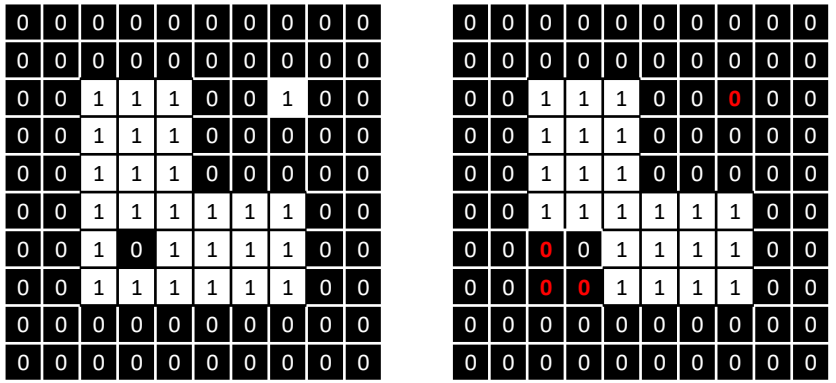
Four basic morphological operations



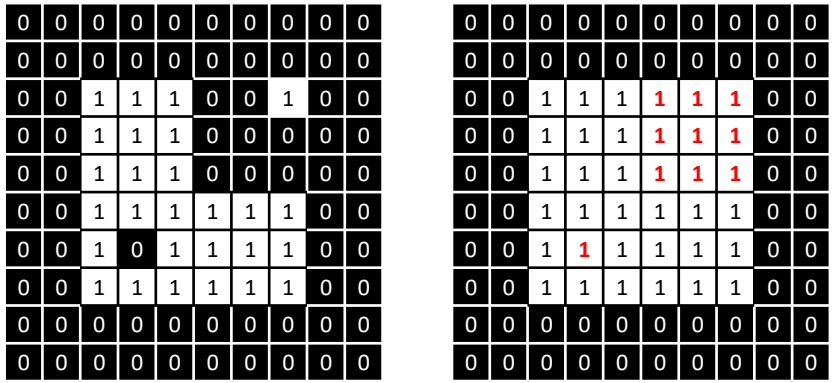
Erosion



Dilation



Opening



Closing

Participate by joining: https://www.sli.do/ #4110

Which morphological operation can lead to the following changes?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	1	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

slido

Which morphological operation can lead to the changes shown on the slide?

ⓘ Start presenting to display the poll results on this slide.

OpenCV examples

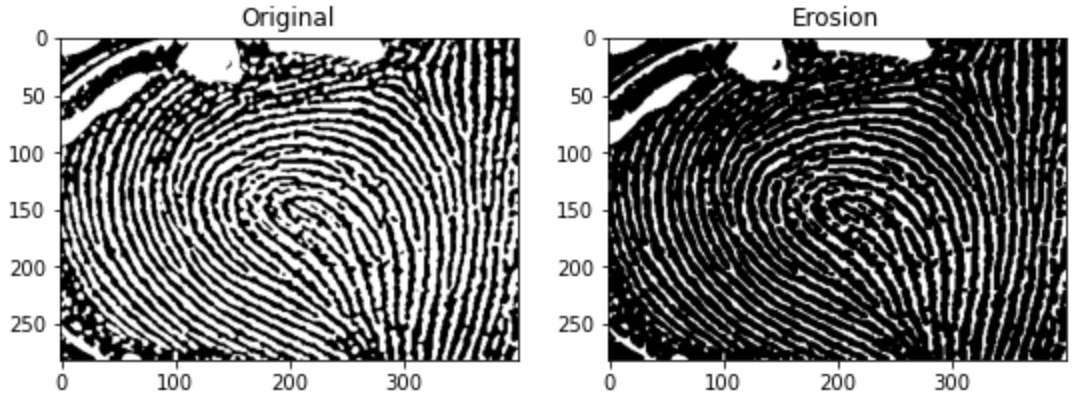
Prepare image and kernel

```
1 img = cv2.imread('fingerprint.png',cv2.IMREAD_GRAYSCALE)
2 kernel = np.ones((3,3), np.uint8)
```

Erosion

```
1 erosion = cv2.erode(img, kernel, iterations = 1)
2 fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
3 ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
4 ax2.imshow(erosion, cmap='gray'), ax2.set_title("Erosion")
```

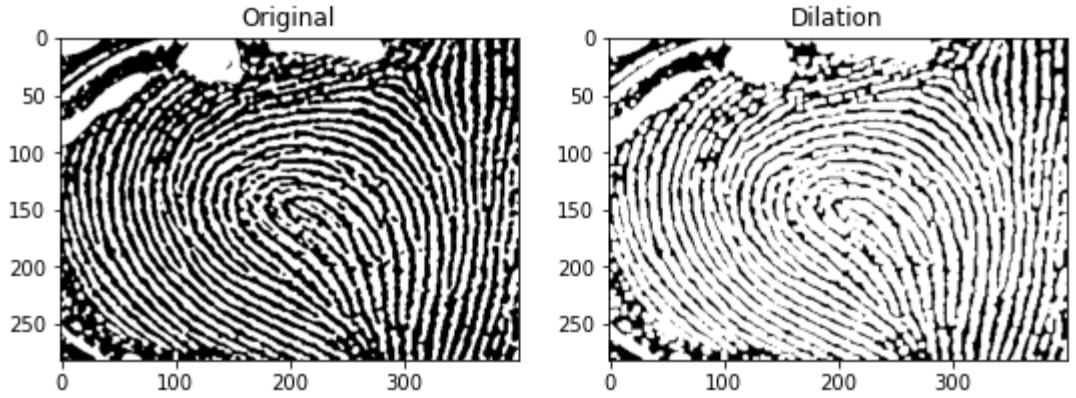
```
(<matplotlib.image.AxesImage at 0x1f96910de08>, Text(0.5, 1.0, 'Erosion'))
```



Dilation

```
1 dilation = cv2.dilate(img, kernel, iterations = 1)
2 fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
3 ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
4 ax2.imshow(dilation, cmap='gray'), ax2.set_title("Dilation")
```

```
(<matplotlib.image.AxesImage at 0x1f9691c5a88>, Text(0.5, 1.0, 'Dilation'))
```

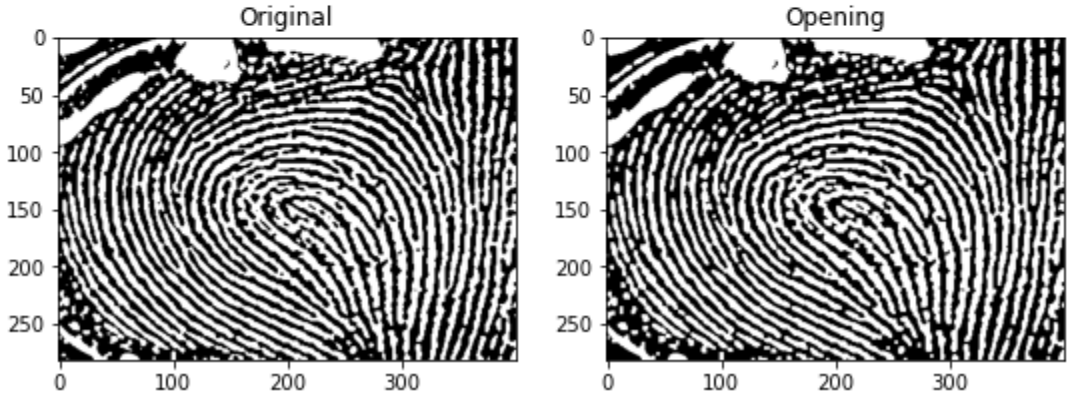


OpenCV examples

Opening

```
1 opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
2 fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
3 ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
4 ax2.imshow(opening, cmap='gray'), ax2.set_title("Opening")
```

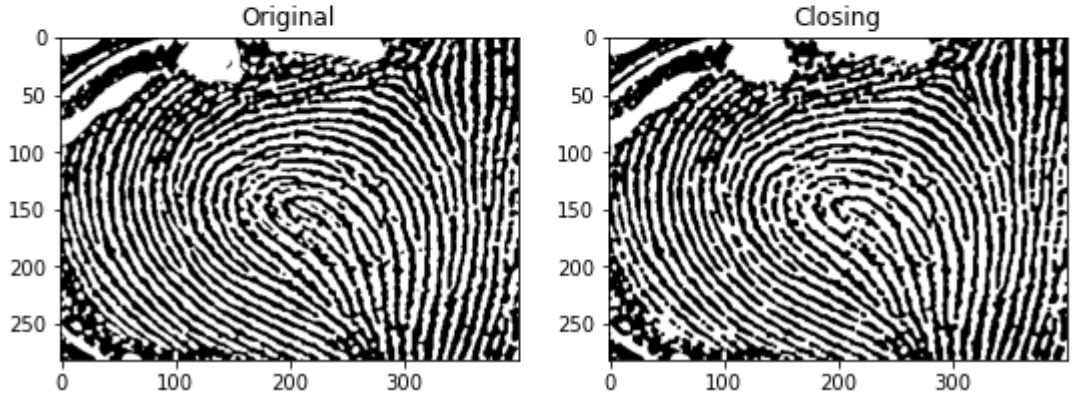
```
(<matplotlib.image.AxesImage at 0x1f96927bf08>, Text(0.5, 1.0, 'Opening'))
```



Closing

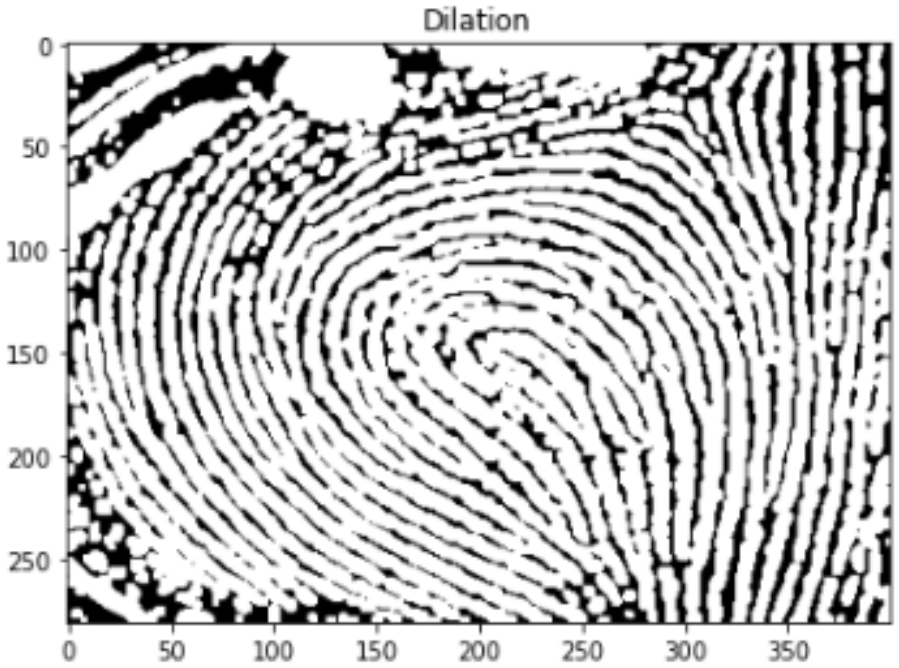
```
1 closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
2 fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
3 ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
4 ax2.imshow(closing, cmap='gray'), ax2.set_title("Closing")
```

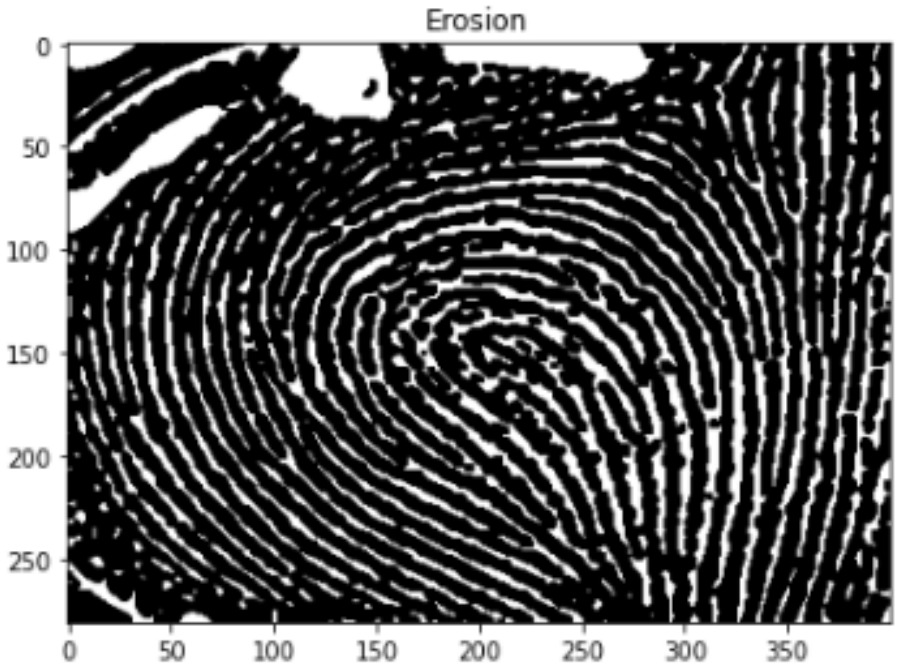
```
(<matplotlib.image.AxesImage at 0x1f9693320c8>, Text(0.5, 1.0, 'Closing'))
```



What other morphological operations can we perform?

- E.g., what would the effect be if we take the difference between Dilation and Erosion?



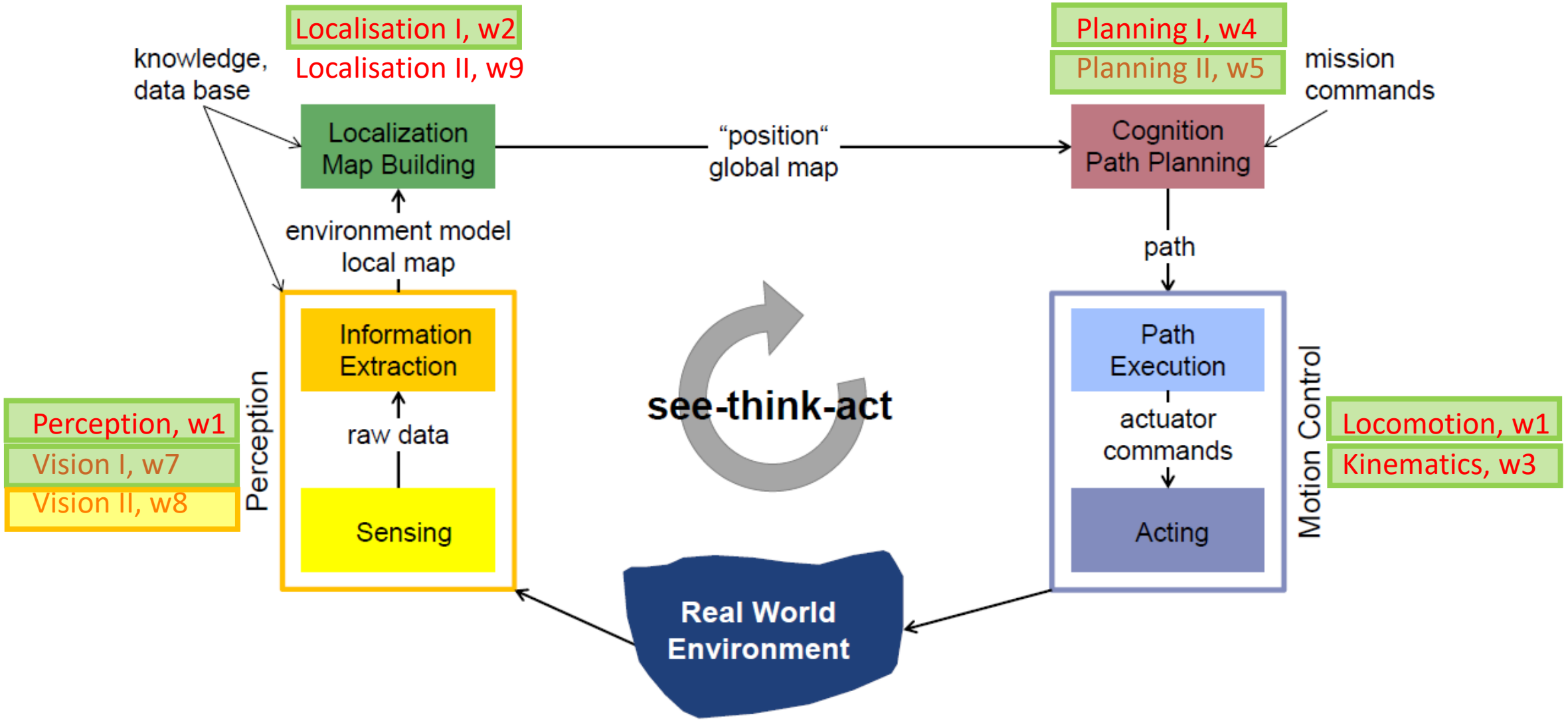


Acknowledgment

- Many slides are adapted from Prof Robert Collins
  - http://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf

- And Prof Peter Corke
  - https://robotacademy.net.au/

Next week: Vision II



R. Siegwart, I. R. Nourbakhsh, D. Scaramuzza. Introduction to autonomous mobile robots. The MIT Press. Second edition. 2011.