MTRN4110 Robot Design
Week 8 – Vision II

Liao “Leo” Wu, Lecturer

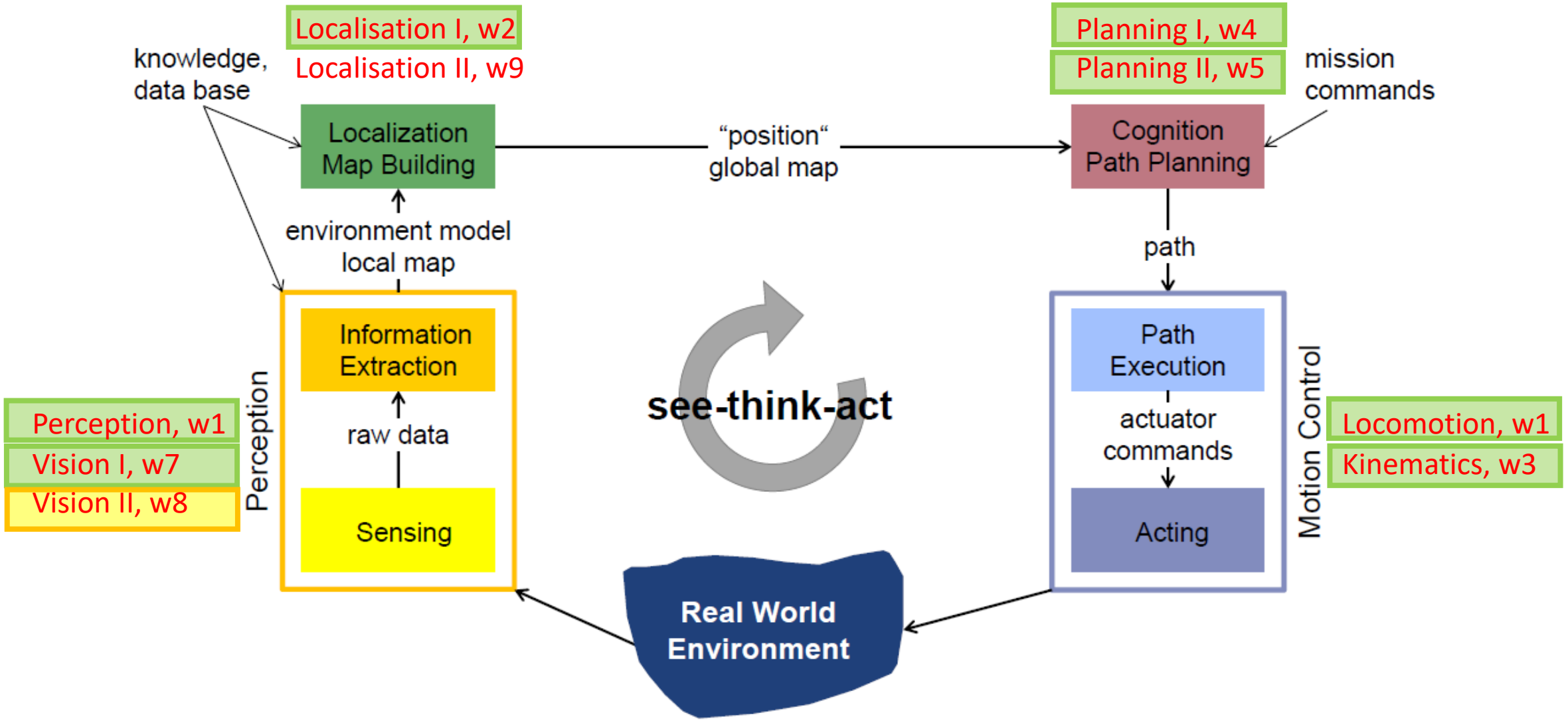
School of Mechanical and Manufacturing Engineering

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

- Feature detection
  - Edge detection
  - Hough transform
  - Contour detection
  - Corner detection
  - SIFT, SURF, and ORB
  - Marker detection
  - Face detection

- Object tracking
  - 8 object tracking algorithms

Feature Detection

Feature detection

- What?
  - Computing abstractions of image information
  - Making local decisions on whether there is a feature of a given type

- Why?
  - Make some "high-level sense" out of a given image (compared with pixel-level)
  - An important step towards understanding the image
    - from image processing to computer vision

- How?
  - Edge detector
  - Contour detector
  - Corner detector
  - SIFT
  - SURF
  - ORB
  - …

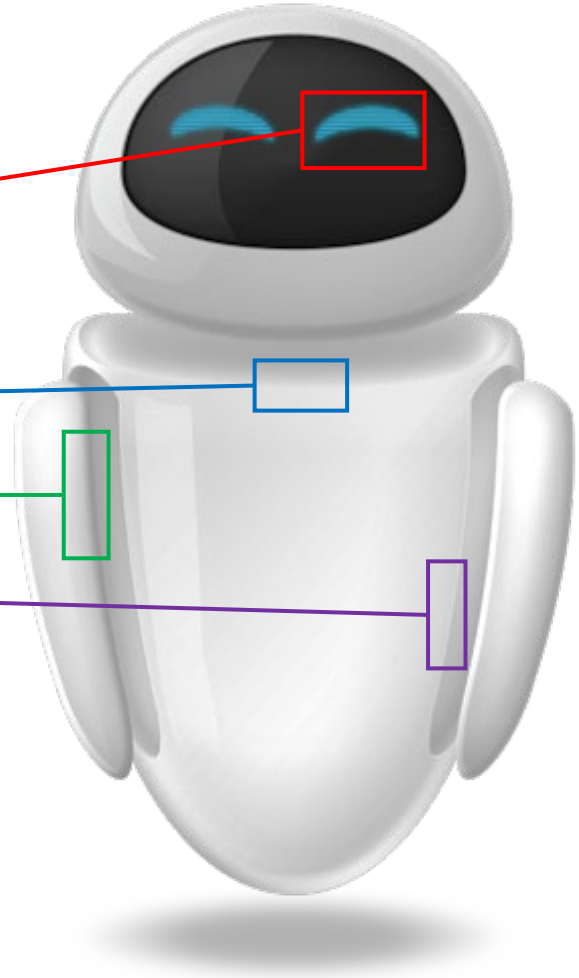
https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)
https://en.wikipedia.org/wiki/Edge_detection#/media/File:%C3%84%C3%A4retuvastuse_n%C3%A4ide.png

Edge Detection

What is an Edge?

- Edge: Sharp change in brightness (discontinuities)

- Origin?
  - Surface colour discontinuity
  - Surface normal discontinuity
  - Depth discontinuity
  - Illumination discontinuity

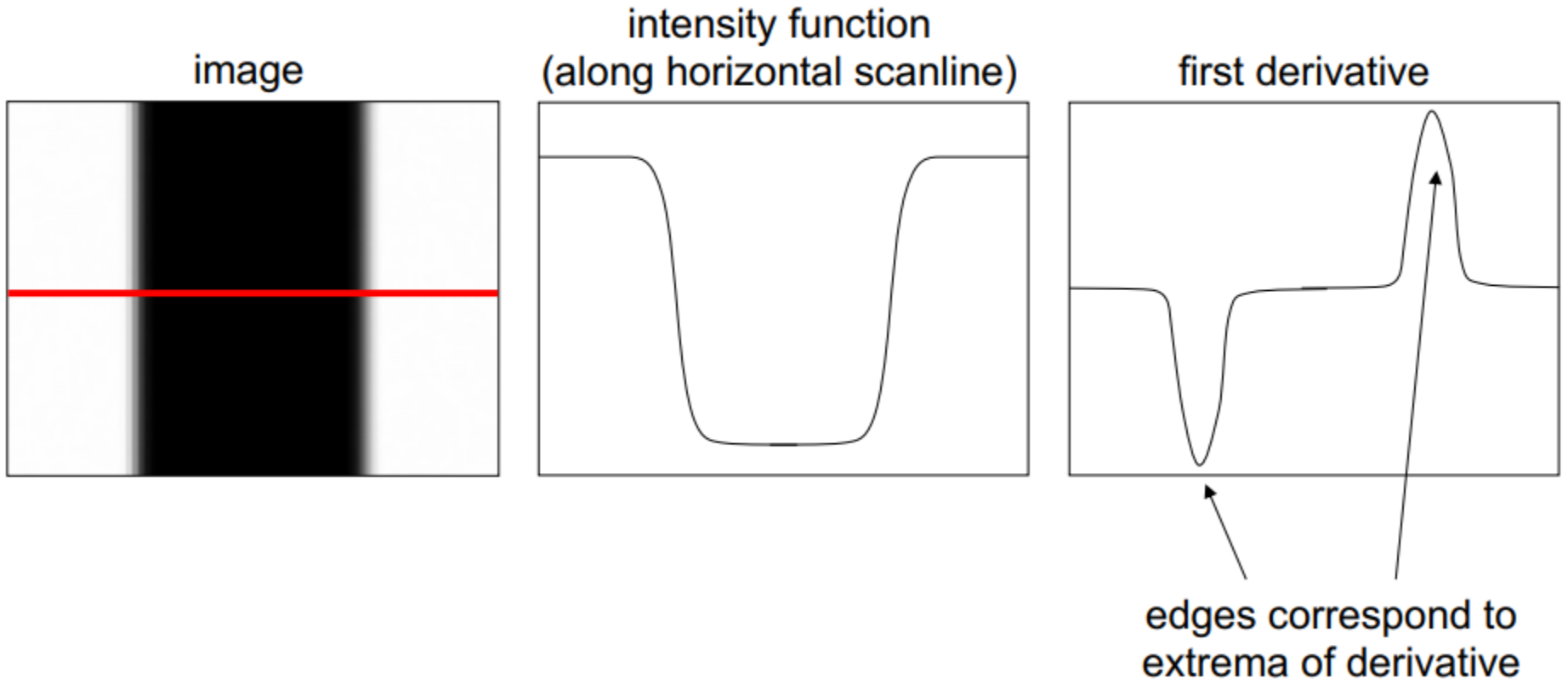


- Edge detection
  - Image processing that finds edges (discontinuities) in images

http://icongal.com/gallery/icon/444/128/robot_eve

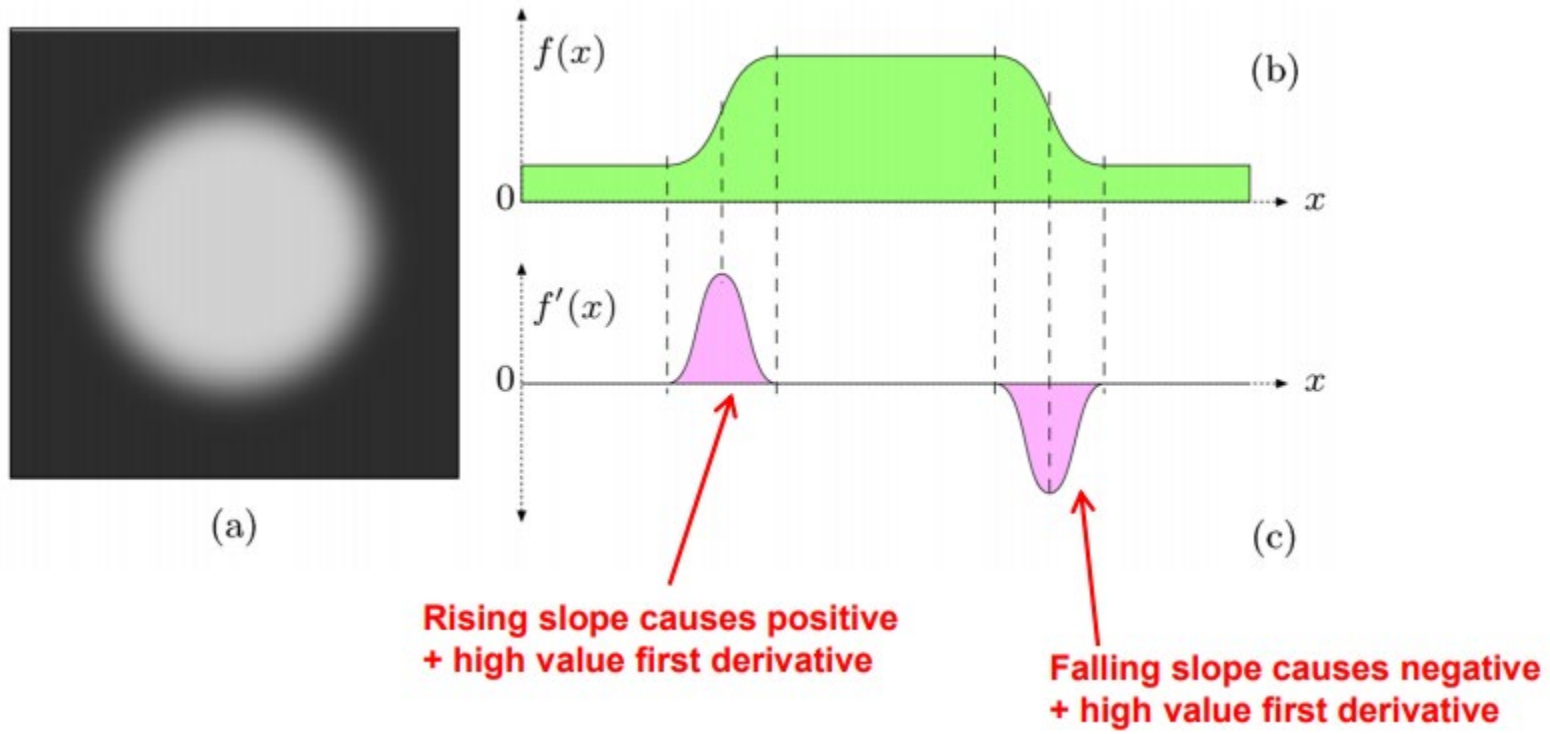
Characteristics of edges

- An edge is a place of rapid change in the image intensity function
- Positive or negative high value first derivative



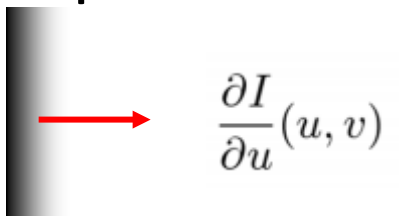
Characteristics of edges

- An edge is a place of rapid change in the image intensity function
- Positive or negative high value first derivative



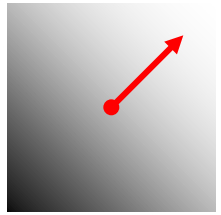
A simple edge detector – Image Gradient

- Image Gradient
  - Image derivatives in horizontal and vertical directions $\frac{\partial I}{\partial u}(u,v)$ and $\frac{\partial I}{\partial v}(u,v)$
  - Image gradient at location (u,v) $\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix}$
  - Gradient magnitude $|\nabla I|(u,v) = \sqrt{\left(\frac{\partial I}{\partial u}(u,v)\right)^2 + \left(\frac{\partial I}{\partial v}(u,v)\right)^2}$
  - Gradient direction $\theta = tan^{-1}\left(\frac{\frac{\partial I}{\partial v}(u,v)}{\frac{\partial I}{\partial u}(u,v)}\right)$
- Compute the Image Gradient



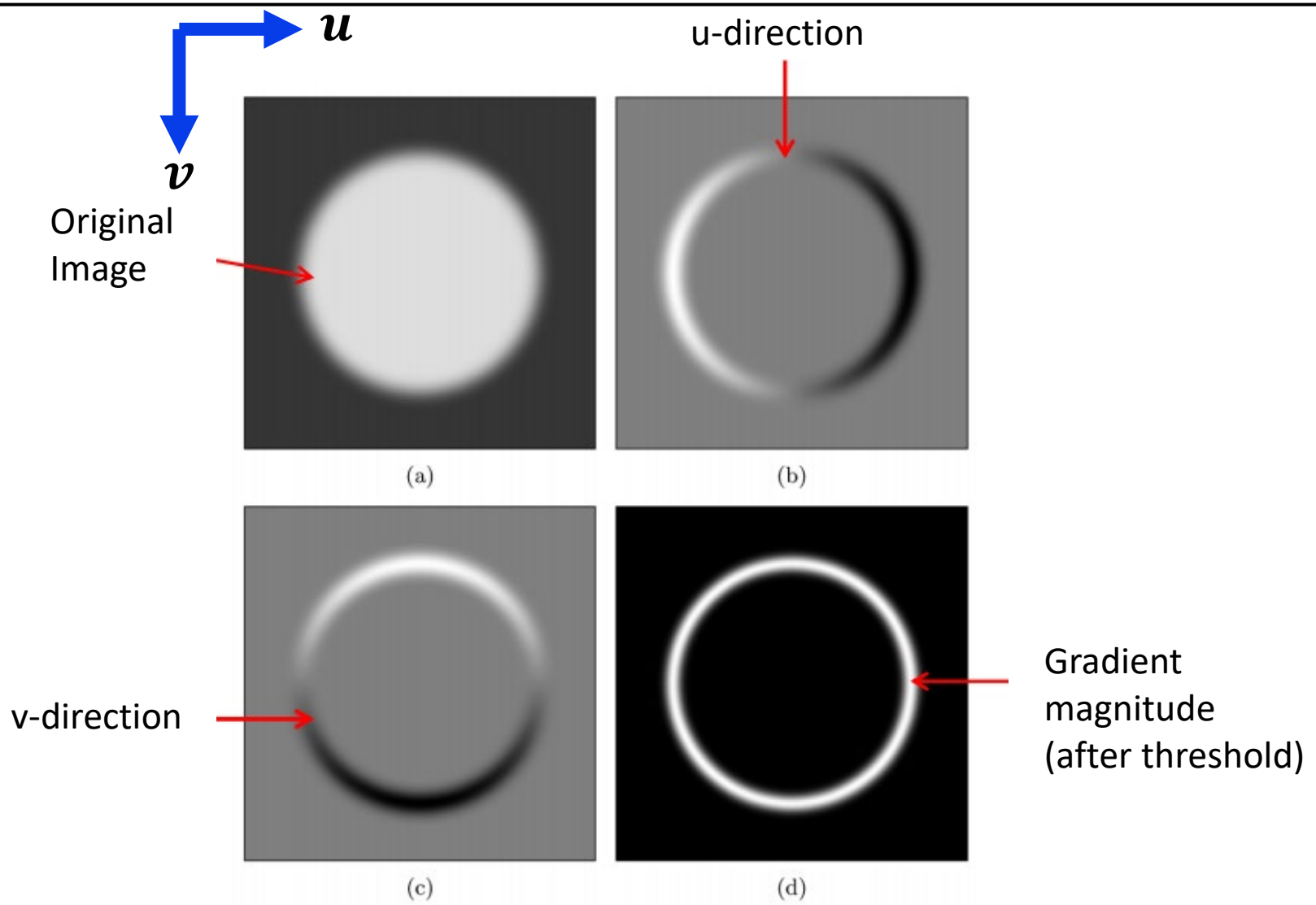
$\frac{\partial I}{\partial u}(u,v)$

$\frac{\partial I}{\partial v}(u,v)$



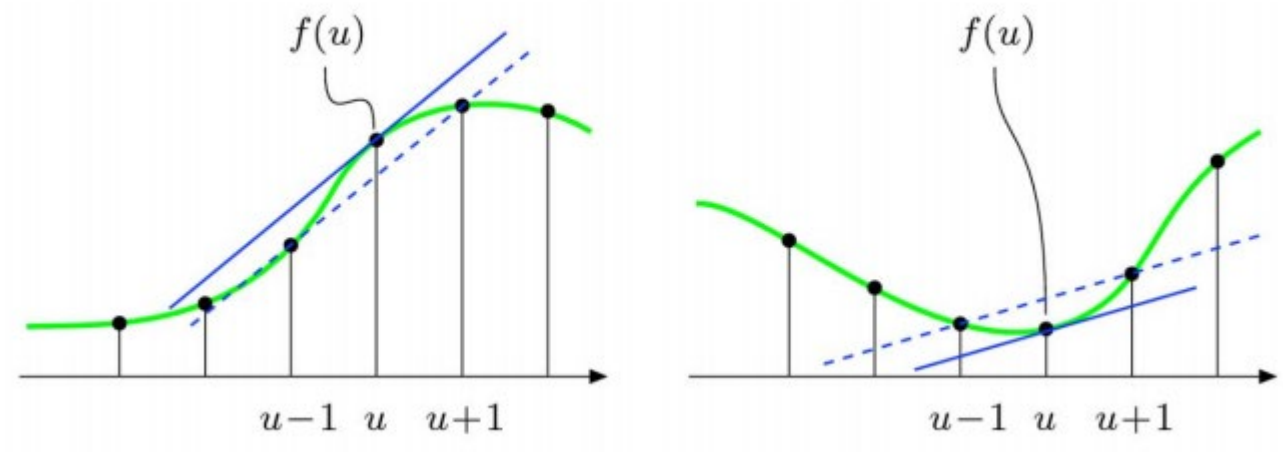
$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I}{\partial u}(u,v) \\ \frac{\partial I}{\partial v}(u,v) \end{bmatrix}$

A simple edge detector - Image Gradient



How to compute the image gradient efficiently?

- Recall that we can compute derivative of discrete function as

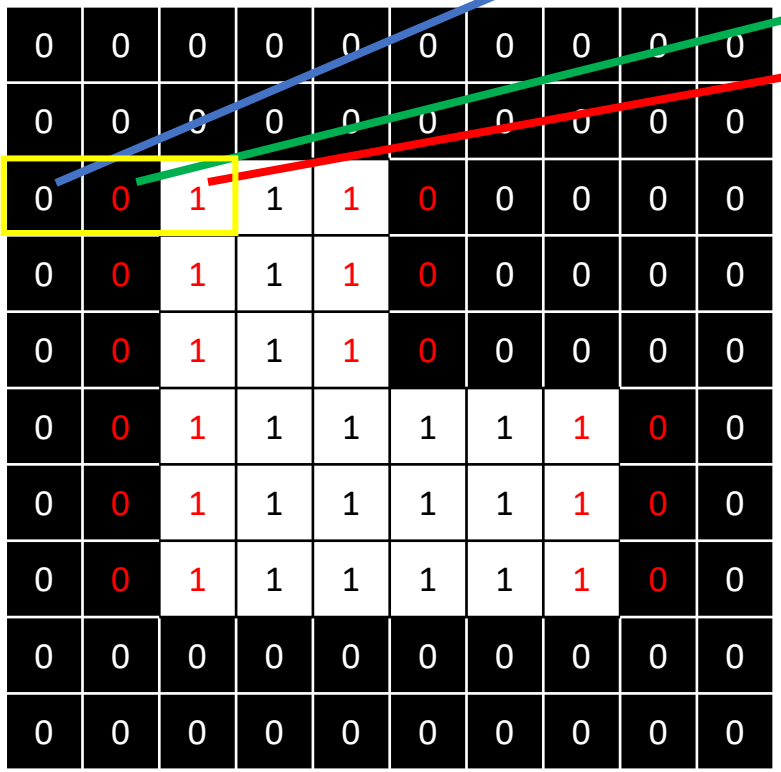


$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot \big(f(u+1) - f(u-1)\big)$$

How to compute the image gradient efficiently?

$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot (f(u+1) - f(u-1))$$

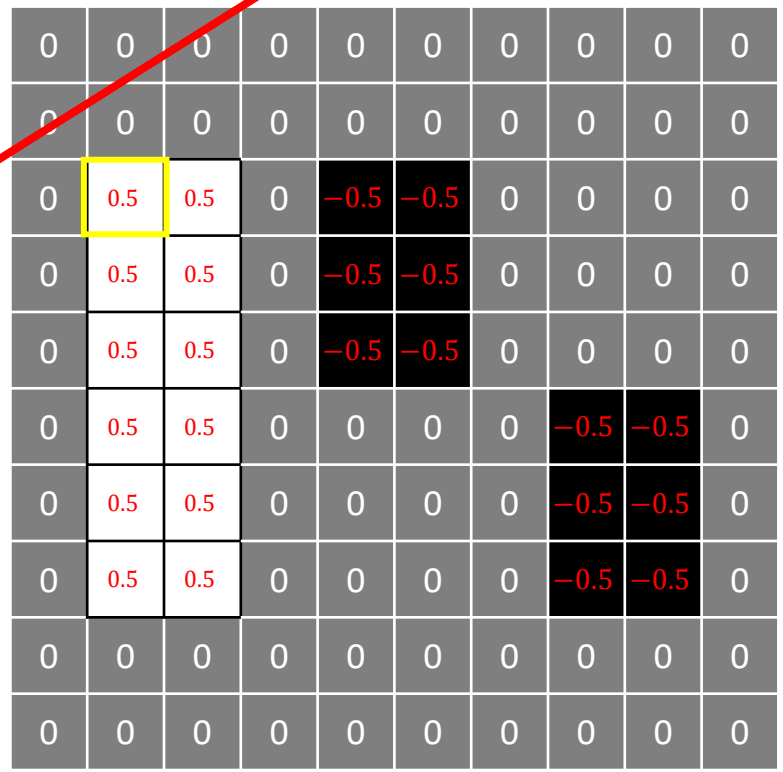
$$I'(u,v) = I(u-1,v) \times 0.5 \times (-1) + I(u,v) \times 0.5 \times (0) + I(u+1,v) \times 0.5 \times (1)$$



$I(u,v)$

$0.5 \times$ | -1 | **0** | 1 |

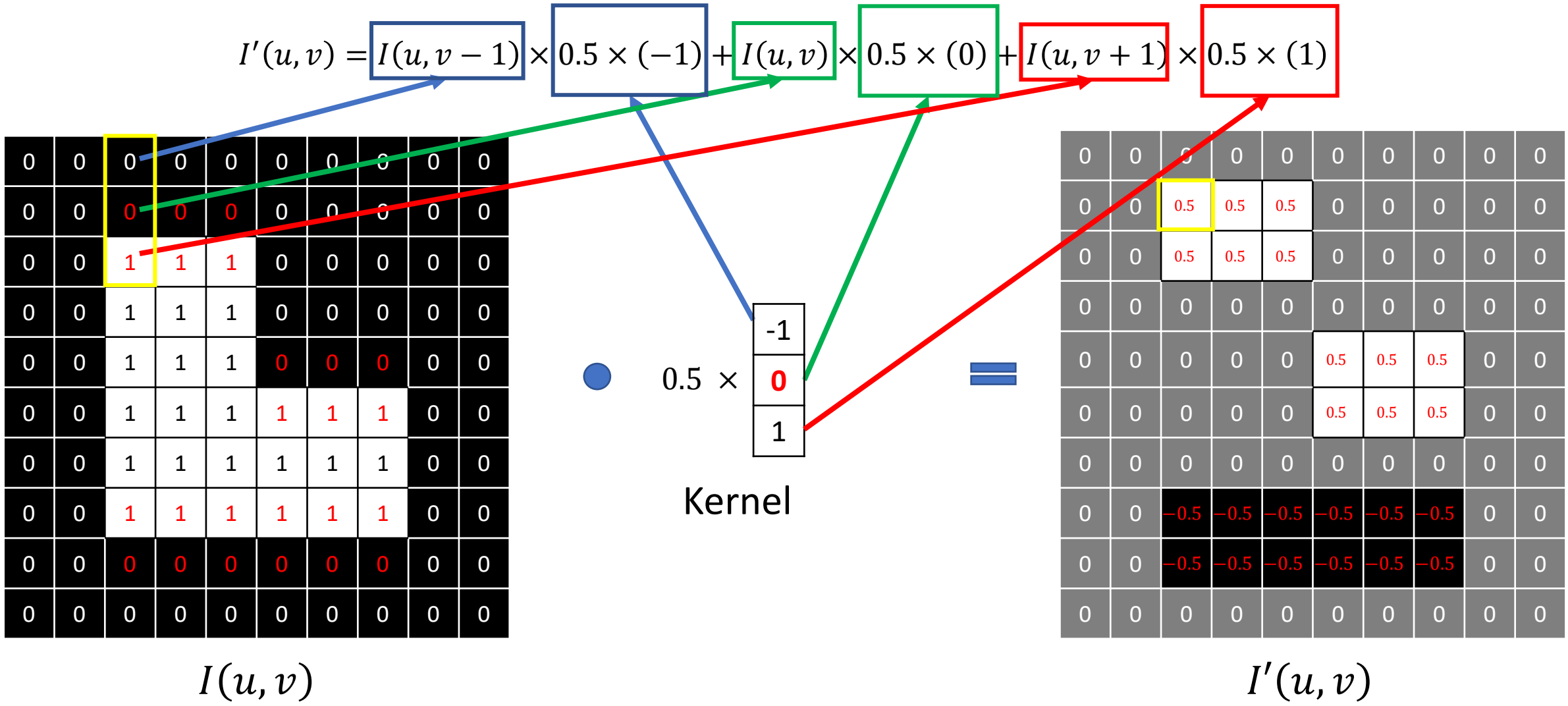
Kernel



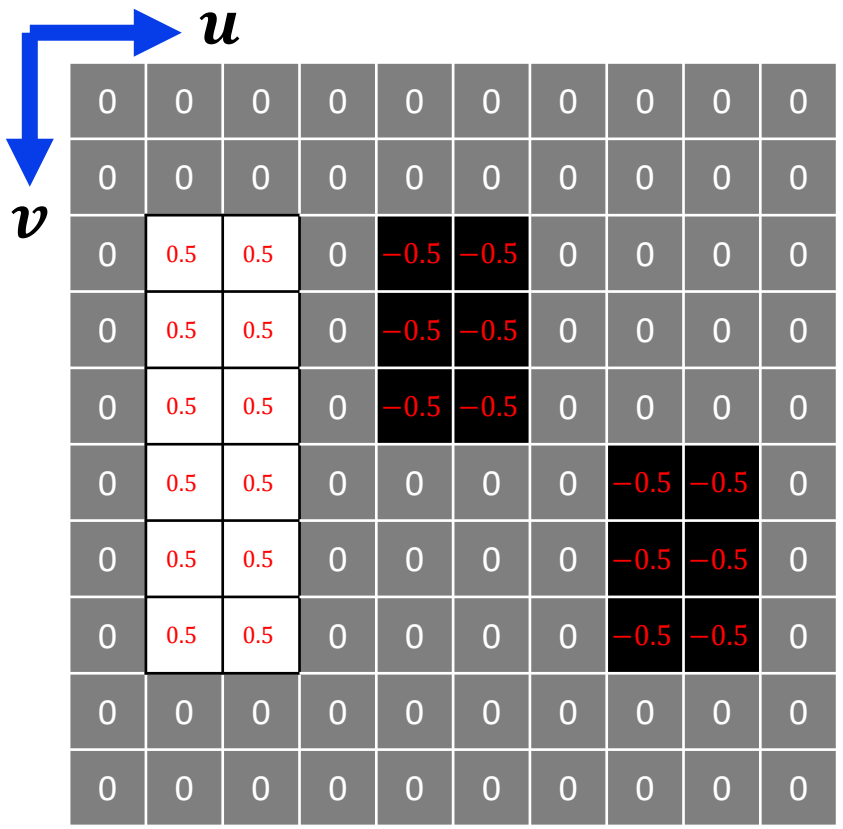
$I'(u,v)$

How to compute the image gradient efficiently?

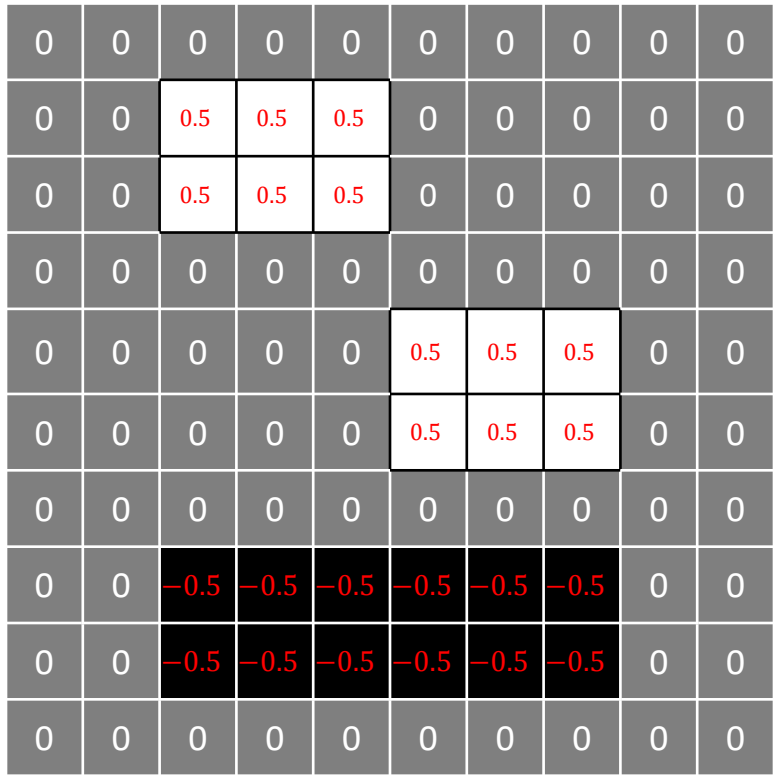
$$\frac{df}{du}(u) \approx \frac{f(u+1) - f(u-1)}{2} = 0.5 \cdot (f(u+1) - f(u-1))$$



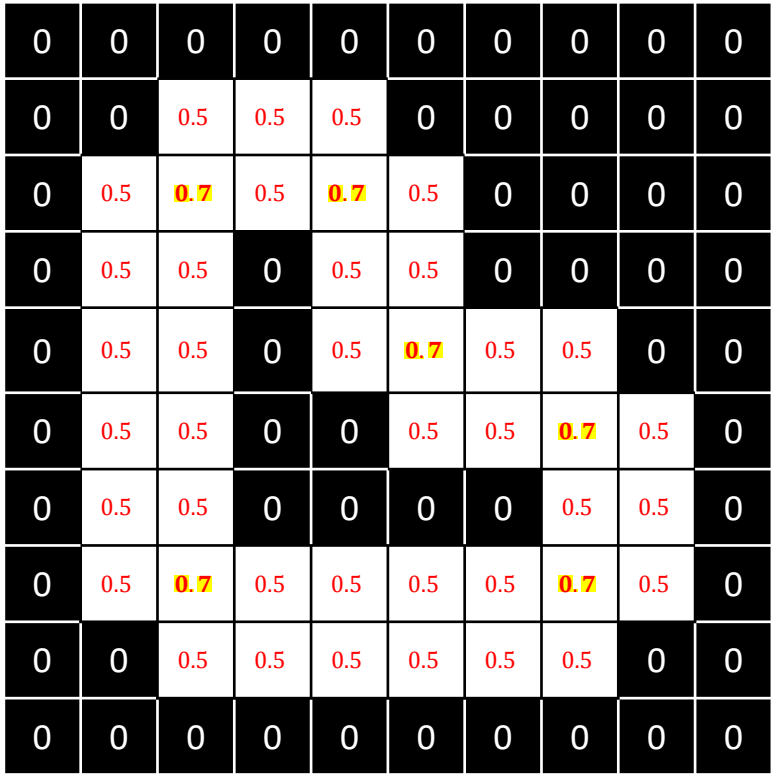
How to compute the image gradient efficiently?



u-direction

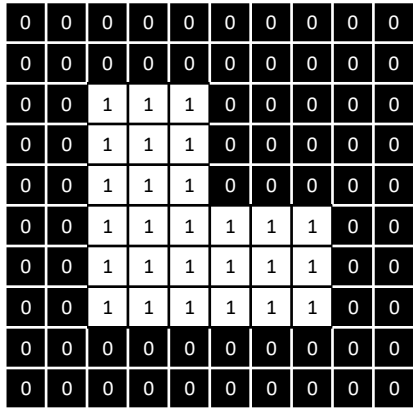


v-direction

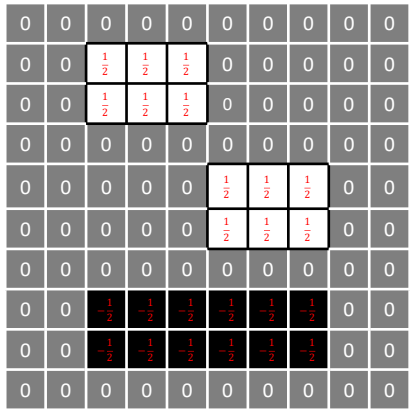


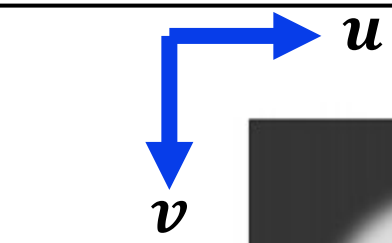
$$|\nabla I|(u,v) = \sqrt{\left(\frac{\partial I}{\partial u}(u,v)\right)^2 + \left(\frac{\partial I}{\partial v}(u,v)\right)^2}$$

A simple edge detector – Finite differentiation filter



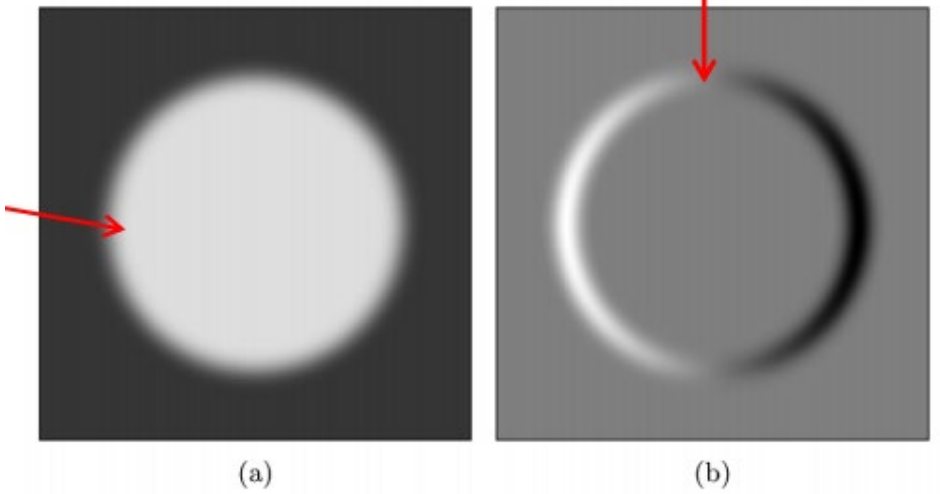
Kernel = $0.5 \times$ | -1 | **0** | 1 |



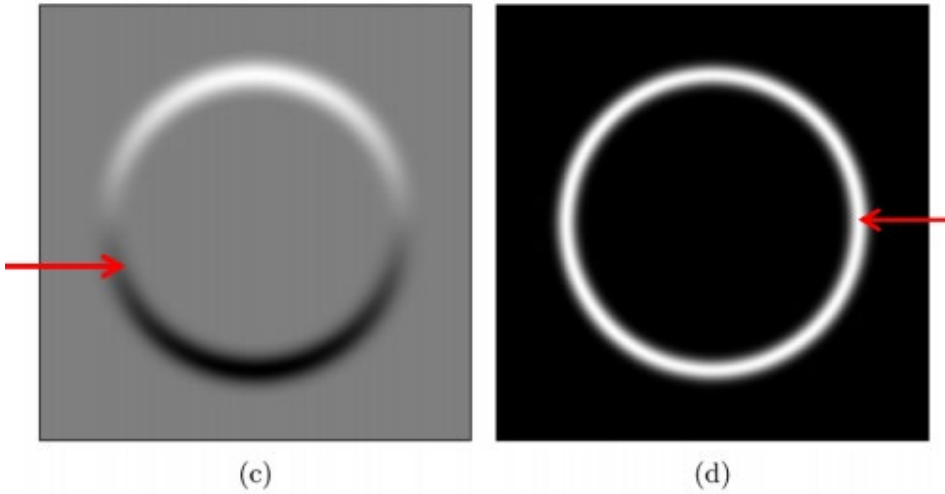


Original Image

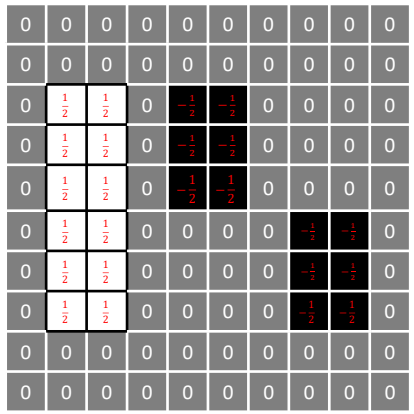
u-direction



v-direction



Kernel = $0.5 \times$ | -1 | **0** | 1 |

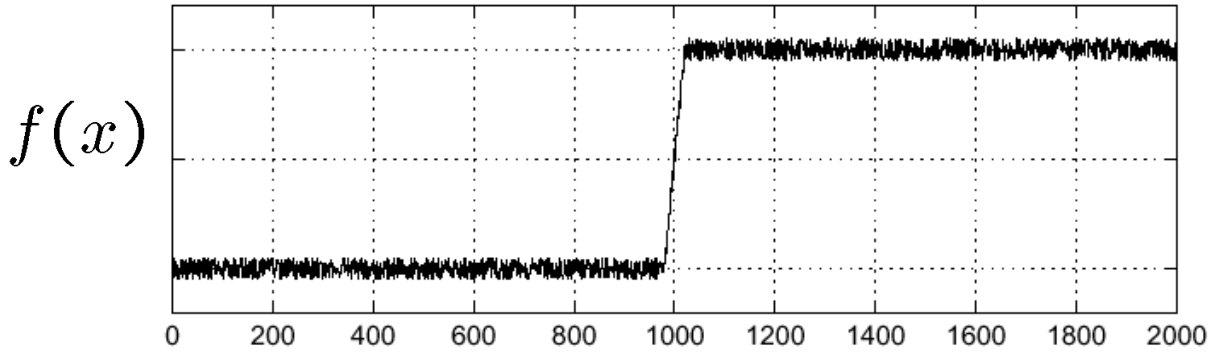


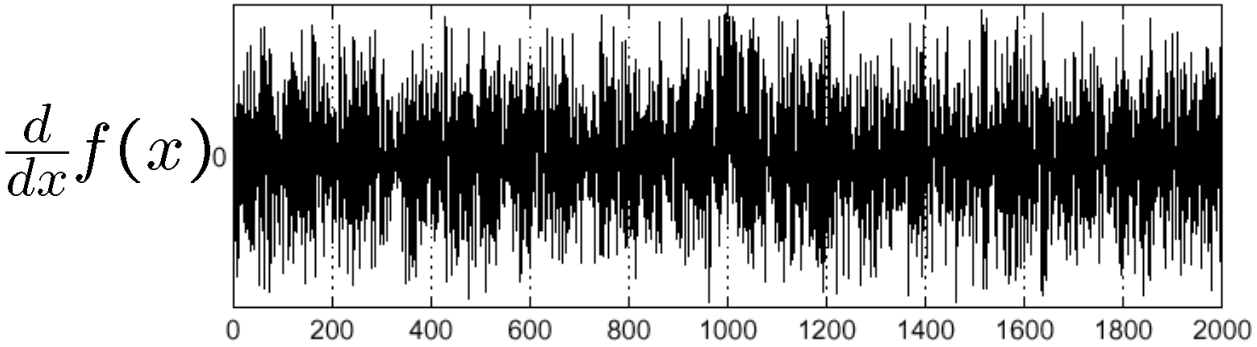
$$|\nabla I|(u,v) = \sqrt{\left(\frac{\partial I}{\partial u}(u,v)\right)^2 + \left(\frac{\partial I}{\partial v}(u,v)\right)^2}$$

Gradient magnitude (after threshold)

Problem - Derivative usually suffers from noise

- Consider a single row or column of the image



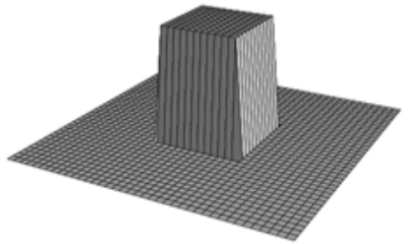


Where is the edge?

Smoothing

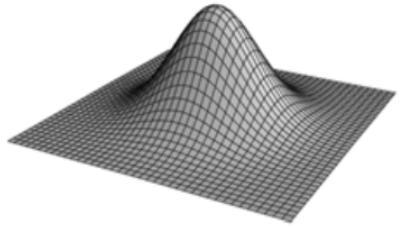
- Finite differentiation filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbours
  - Generally, the larger the noise, the stronger the response

- Smoothing the image to reduce noise before computing the derivative
  - Smoothing is a filter operation that averages a pixel using its surrounding pixels
  - Can be performed by using a kernel filter similarly as introduced previously



$$\frac{1}{3} \times \begin{bmatrix} 1 & \mathbf{1} & 1 \end{bmatrix} \qquad \frac{1}{3} \times \begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix}$$

Mean Smoothing



$$\frac{1}{4} \times \begin{bmatrix} 1 & \mathbf{2} & 1 \end{bmatrix} \qquad \frac{1}{4} \times \begin{bmatrix} 1 \\ \mathbf{2} \\ 1 \end{bmatrix}$$

Gaussian Smoothing

Smoothing with different filters

	Mean	Gaussian
3x3		
5x5		
7x7		

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\frac{1}{25} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\frac{1}{49} \times \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

1/16

1	2	1
2	4	2
1	2	1

1/273

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

1/1003

0	0	1	2	1	0	0
0	3	13	22	13	3	0
1	13	59	97	59	13	1
2	22	97	159	97	22	2
1	13	59	97	59	13	1
0	3	13	22	13	3	0
0	0	1	2	1	0	0

Sobel filter

- Presented by Irwin Sobel and Gary Feldman in 1968.
- Uses two 3 x 3 kernels
- One for horizontal changes, and one for vertical

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Magnitude

$$\mathbf{G} = \sqrt{\mathbf{G}_x{}^2 + \mathbf{G}_y{}^2}$$

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

Sobel filter

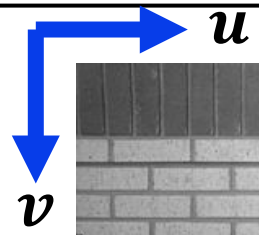
- Combination of Gaussian Smoothing and Finite Differentiation

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} +1 & 0 & -1 \end{bmatrix}$$

Gaussian smoothing

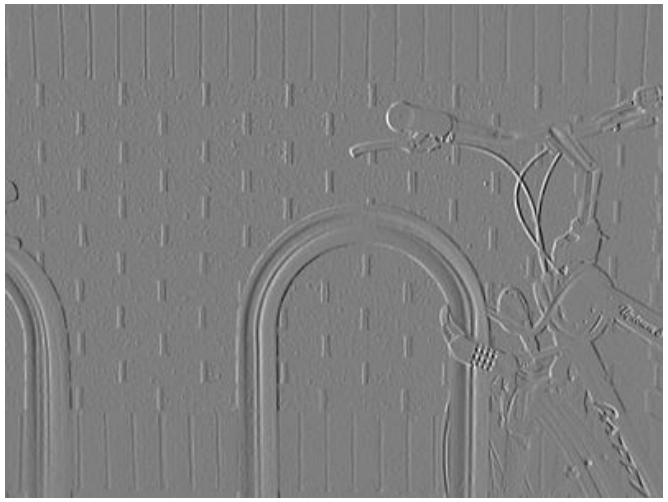
Finite differentiation

Sobel filter

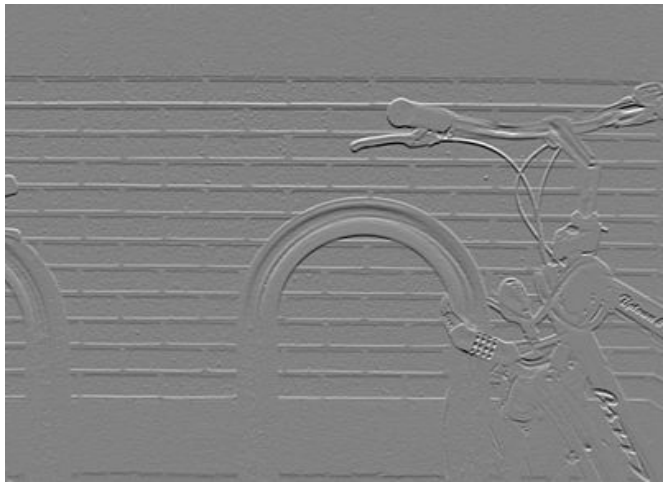


Original Image

u-direction



v-direction



Gradient magnitude (after threshold)

Sobel filter – Example in x direction

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

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

$$\bullet\ \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} =$$

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

Sobel filter – Example in x direction

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

$$\bullet\ \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} =$$

0	0	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	0	0	0	0

Sobel filter – Example in x direction

What should the value be for the yellow cell on the right?

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

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

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} =$$

0	0	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	0	0	0	0

slido

What should the value be for the yellow cell on the right?

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

Sobel filter – Example in x direction

What should the value be for the yellow cell on the right?

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

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

$$\bullet\ \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} =$$

0	0	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	0	0	0	0

Sobel filter – Example in x direction

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

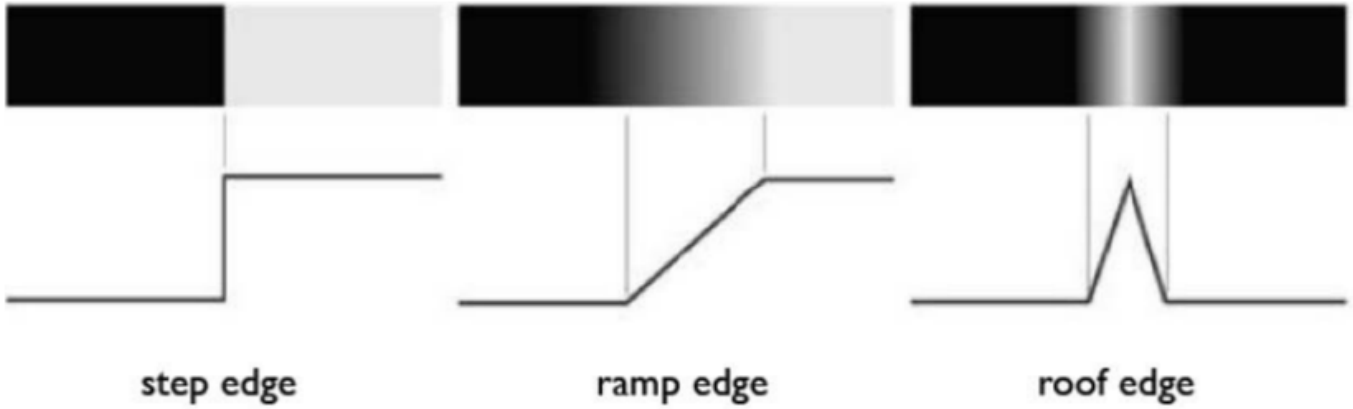
1	0	-1
2	0	-2
1	0	-1

$$\bullet\ \mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} =$$

0	0	0	0	0	0	0	0	0	0
0	-1	-1	0	1	1	0	0	0	0
0	-3	-3	0	3	3	0	0	0	0
0	-4	-4	0	4	4	0	0	0	0
0	-4	-4	0	3	3	0	1	1	0
0	-4	-4	0	1	1	0	3	3	0
0	-4	-4	0	0	0	0	4	4	0
0	-3	-3	0	0	0	0	3	3	0
0	-1	-1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	0	0	0

Sobel filter problems

- Poor localisation
  - Trigger response in multiple adjacent pixels (Consider grayscale image instead of binary image)



- Thresholding value favours certain directions over others
  - Can miss oblique edges more than horizontal or vertical edges

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad \mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Canny edge detector

- Developed by John F. Canny in 1986.
- Probably the most widely used edge detector in computer vision
- A multi-stage algorithm operating at different scales:
  1. Filter image with a Gaussian filter
  2. Find the intensity gradient of the image using Sobel filters
  3. Non-maximum suppression:
     - Thin multi-pixel wide "ridges" down to single pixel width
  4. Thresholding and linking (hysteresis):
     - Define two thresholds: low and high
     - Use the high threshold to start edge curves and the low threshold to continue them

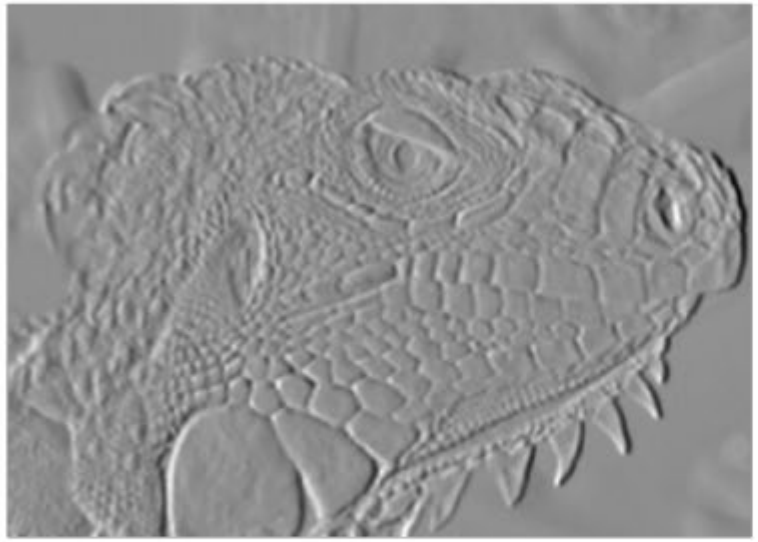
https://en.wikipedia.org/wiki/Canny_edge_detector

Canny edge detector

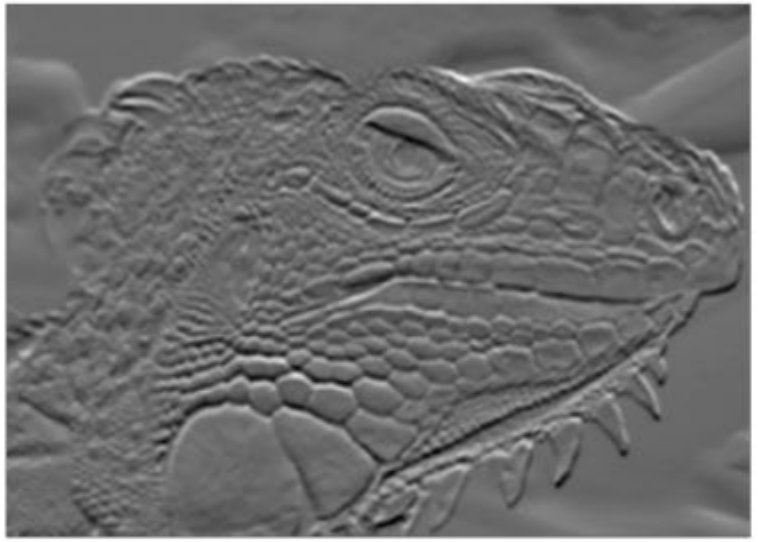
- Original image

Canny edge detector

1. Filter image with a Gaussian filter
2. Find the intensity gradient of the image using Sobel filters



X-Derivative of Gaussian

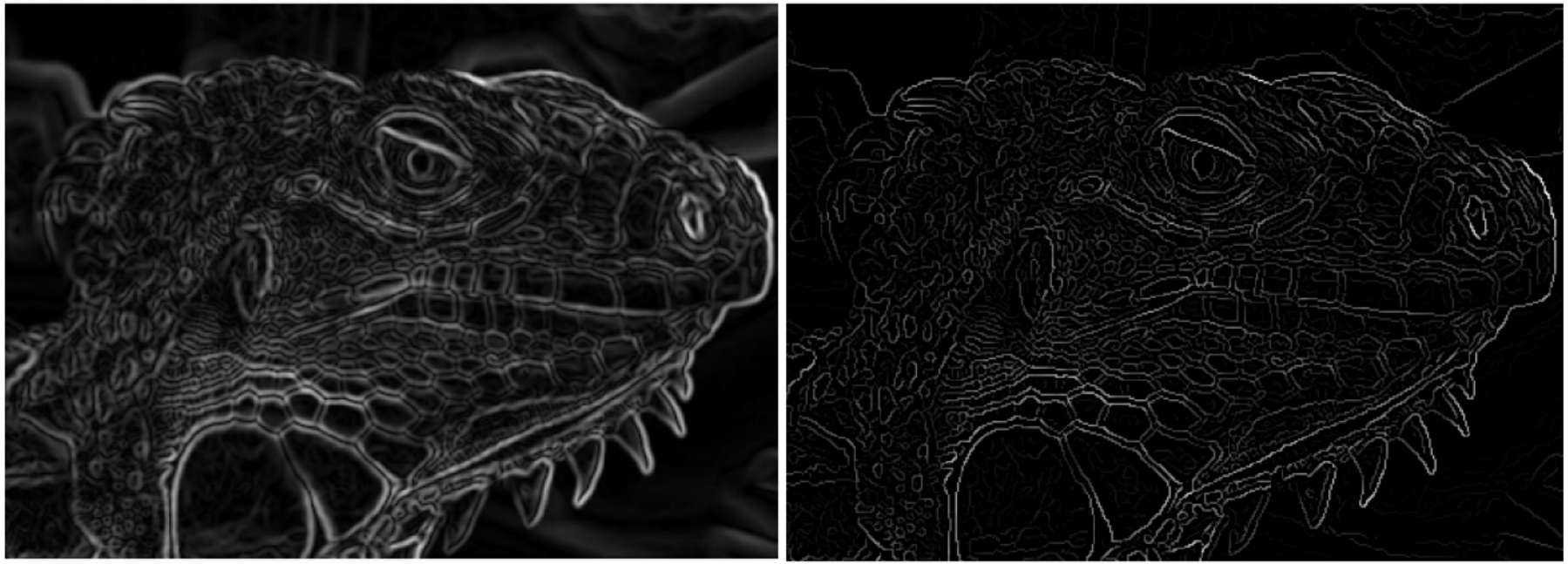


Y-Derivative of Gaussian

Gradient Magnitude

Canny edge detector

3. Non-maximum suppression:
   - Thin multi-pixel wide "ridges" down to single pixel width



Before

After

Canny edge detector

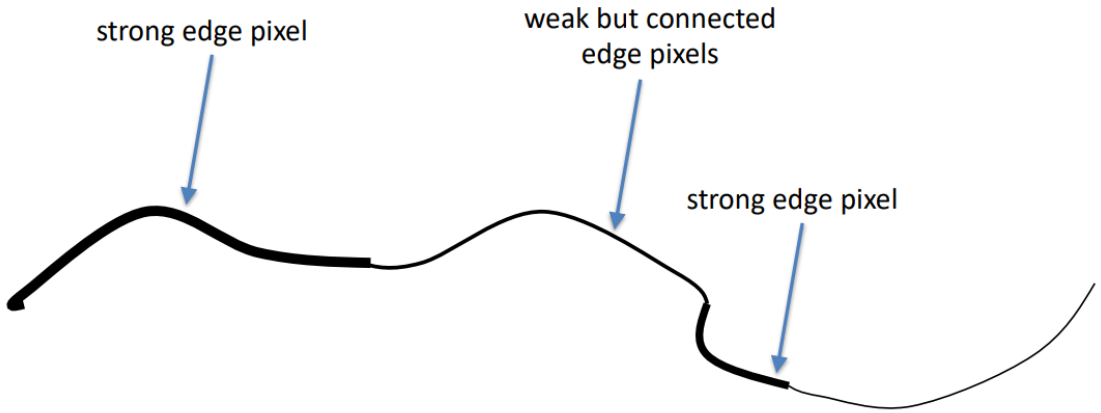
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
     - If less than Low, not an edge
     - If greater than High, strong edge
     - If between Low and High, weak edge

Thresholding (high)

Thresholding (low)

Canny edge detector

4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
     - If less than Low, not an edge
     - If greater than High, strong edge
     - If between Low and High, weak edge
   - Use the high threshold to start edge curves and the low threshold to continue them
     - For weak edges:
       - Consider its neighbours iteratively then declare it an “edge pixel” if it is connected to an ‘strong edge pixel’ directly or via pixels between Low and High



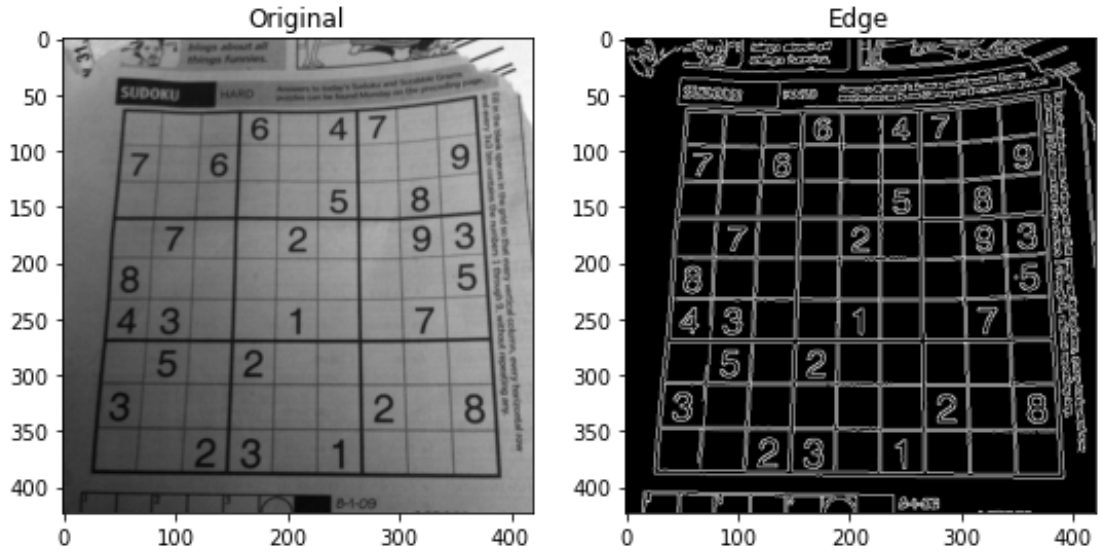
Canny edge detector

Final edges:

OpenCV examples – Edge detection

```python
#import required libraries
import cv2 # OpenCV library
import numpy as np # Numpy library for scientific computing
import matplotlib.pyplot as plt # Matplotlib library for plotting

#canny edge detection
img = cv2.imread('sudoku.jpg',cv2.IMREAD_GRAYSCALE)
edges = cv2.Canny(img,50,100)

#plot the image
fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
ax2.imshow(edges, cmap='gray'), ax2.set_title("Edge")
```

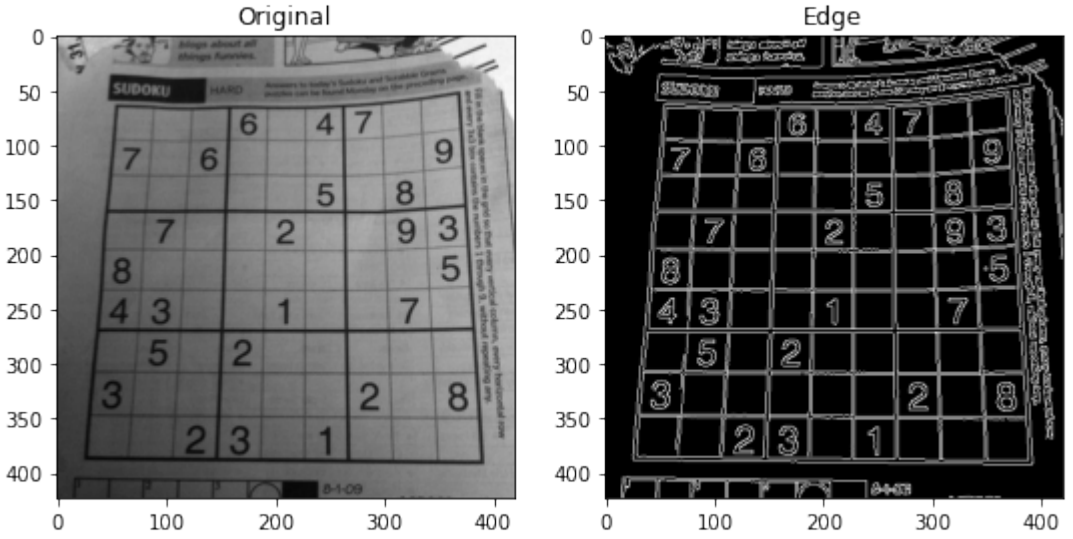
```
(<matplotlib.image.AxesImage at 0x1b383c02b88>, Text(0.5, 1.0, 'Edge'))
```



Hough Transform

Now we have the edges, can we extract lines out of the edge map?

- Hough Transform (HT) can help with this.
- Introduced in 1962 by Paul Hough
- We will introduce line detection using Hough transform, however, note that it can be used for other feature detections such as circle or ellipses detection.

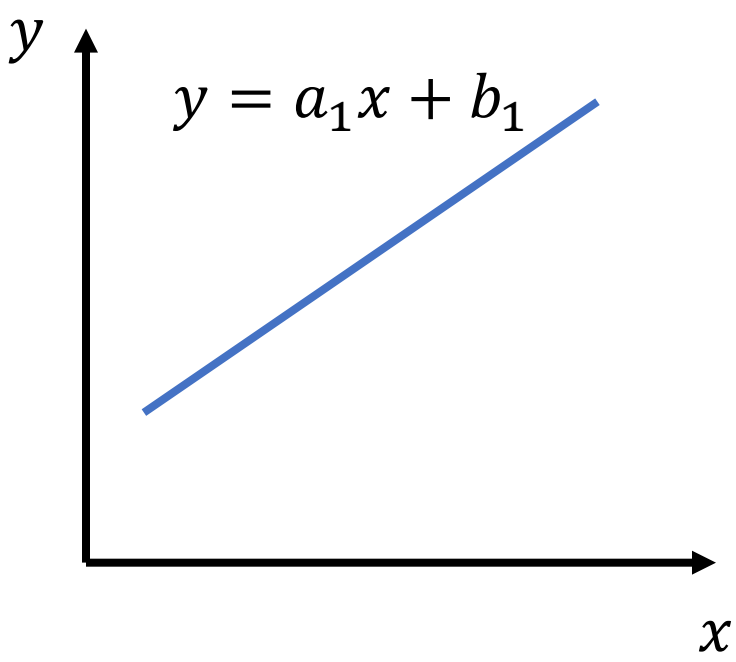


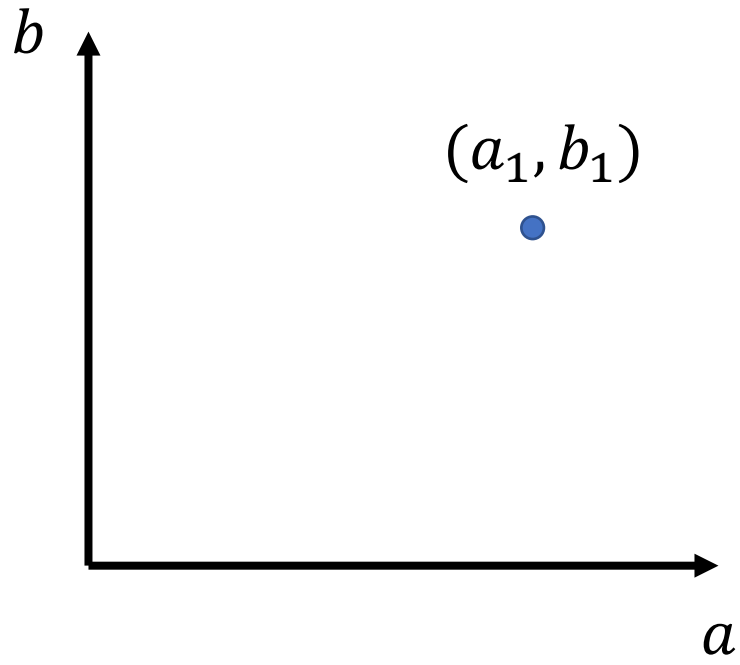
https://en.wikipedia.org/wiki/Hough_transform

Parameter space

- A line can be represented by an infinite set of points (x_i, y_i)
- A line can also be represented by a pair of parameters (a_i, b_i)

A line in the "x-y" space gives a point in the "a-b" space;

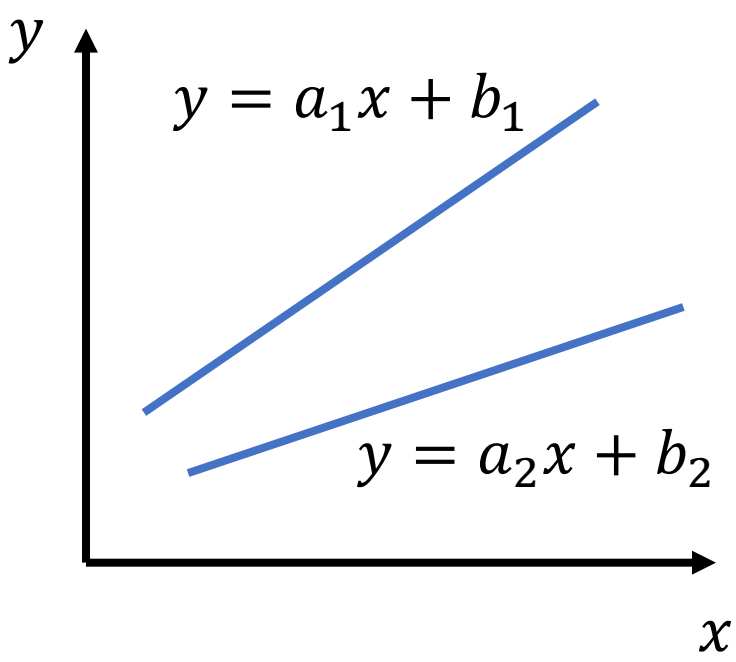


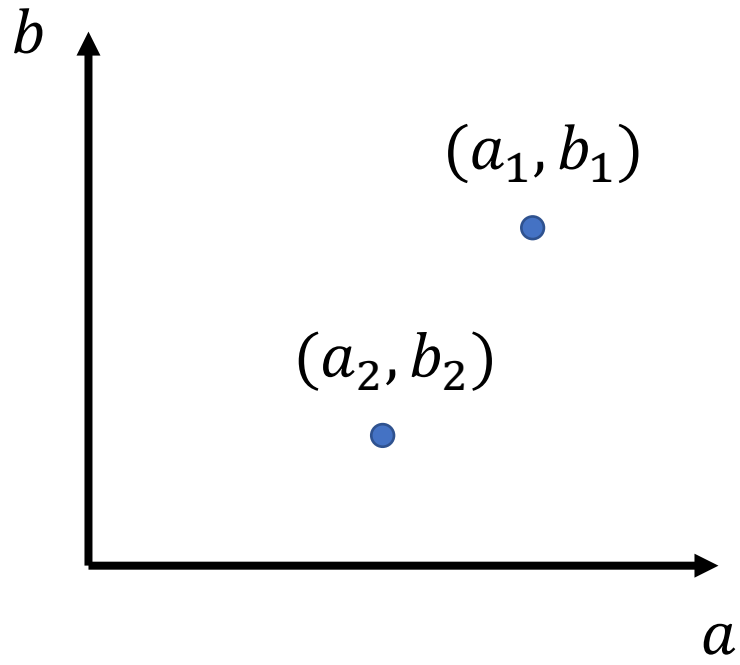


Parameter space

- A line can be represented by an infinite set of points (x_i, y_i)
- A line can also be represented by a pair of parameters (a_i, b_i)

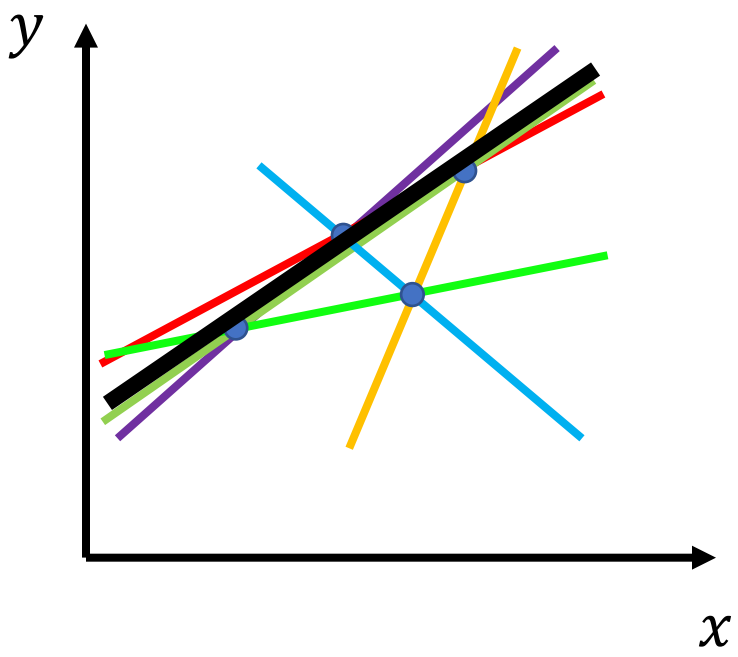
A line in the "x-y" space gives a point in the "a-b" space;

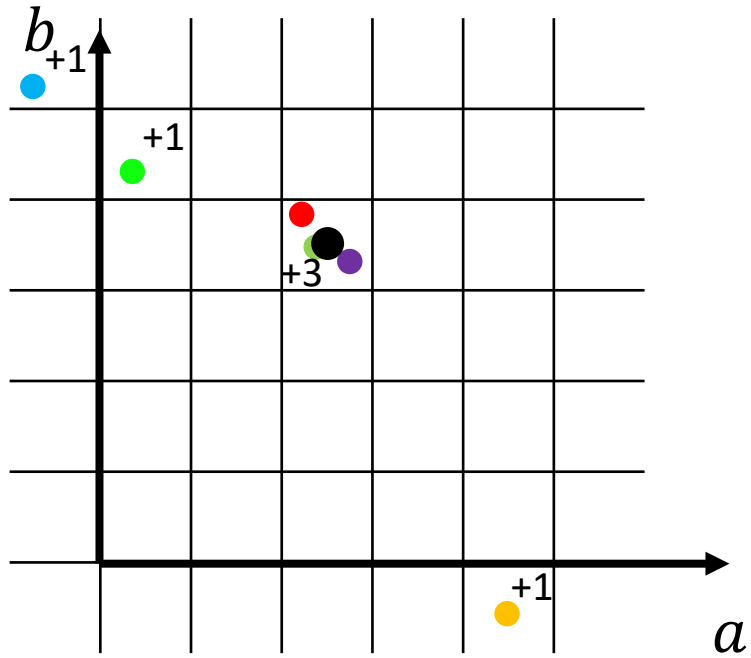




Hough line detection – Version 1

- 1. Quantize the parameter space “a-b” by dividing it into cells
- 2. For each pair of points (x_1, y_1) and (x_2, y_2) detected as an edge, find the point (a, b) in the “a-b” space
- 3. Increase the value of a cell if a point (a, b) belongs to this cell
- 4. Cells receiving more than a certain number of votes are assumed to correspond to lines in the “x-y” space

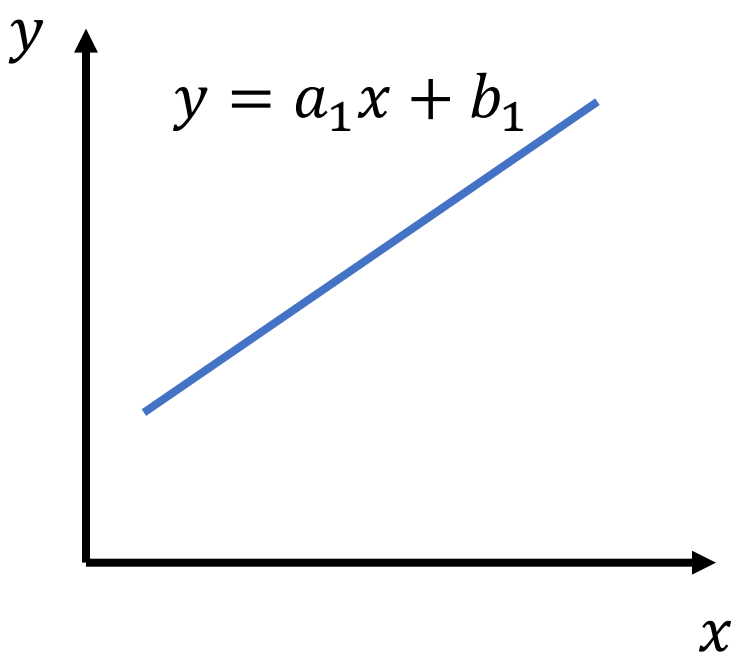


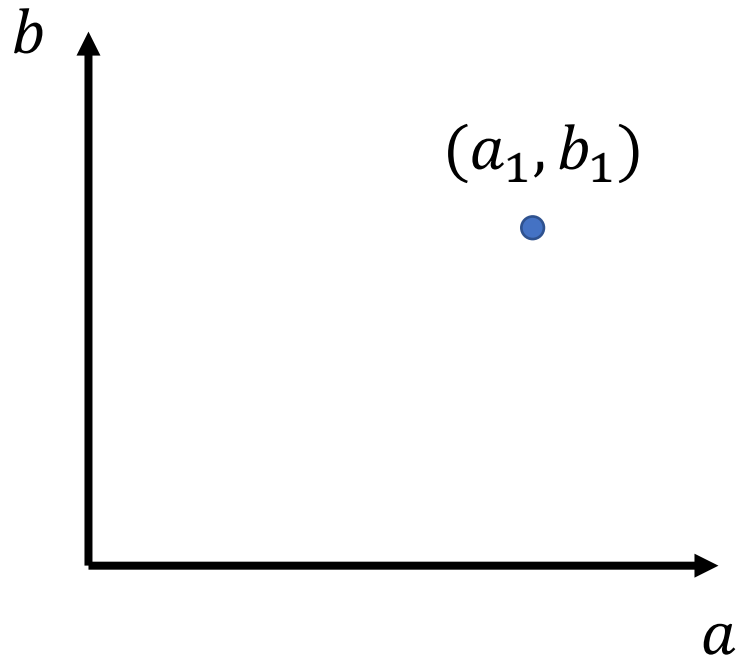


Parameter space

- A line can be represented by an infinite set of points (x_i, y_i)
- A line can also be represented by a pair of parameters (a_i, b_i)

A line in the "x-y" space gives a point in the "a-b" space;



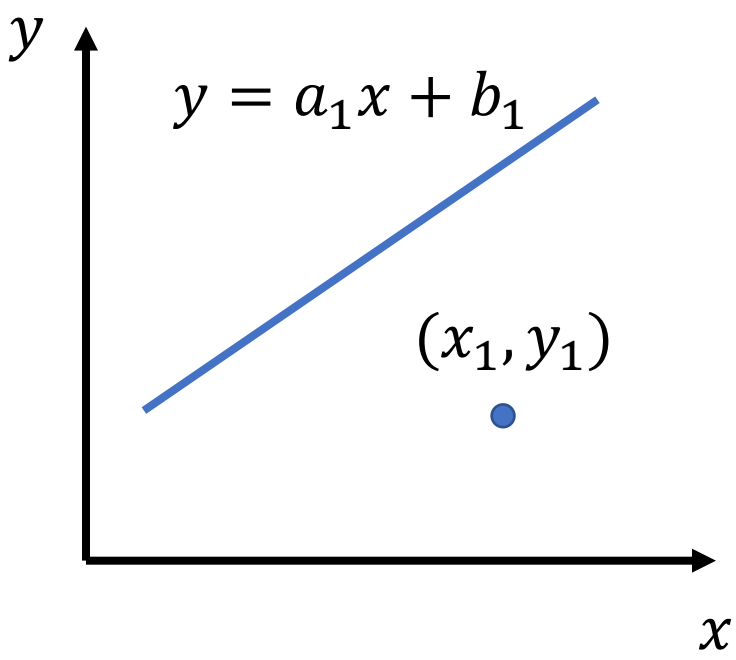


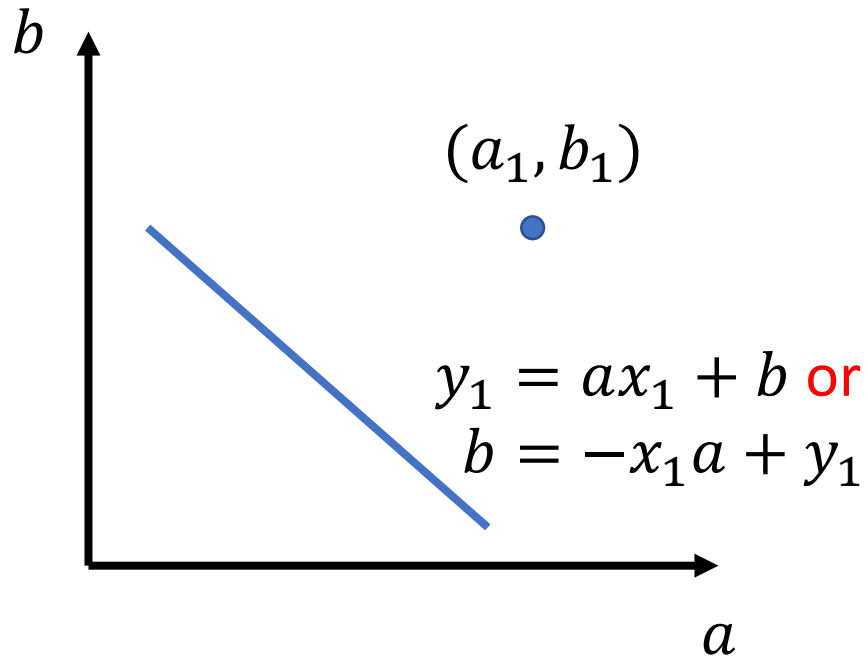
Parameter space

- A line can be represented by an infinite set of points (x_i, y_i)
- A line can also be represented by a pair of parameters (a_i, b_i)

A line in the "x-y" space gives a point in the "a-b" space;

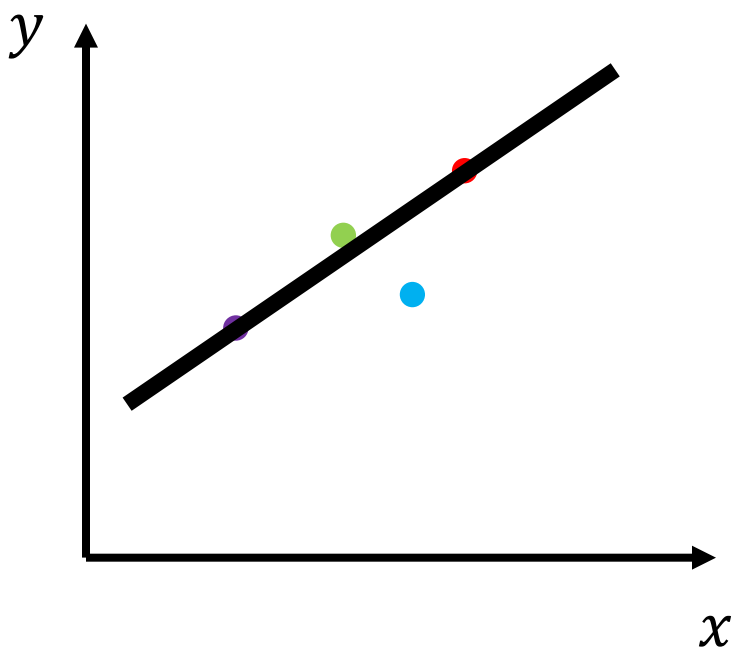
A point in the "x-y" space also corresponds to a line in the "a-b" space.

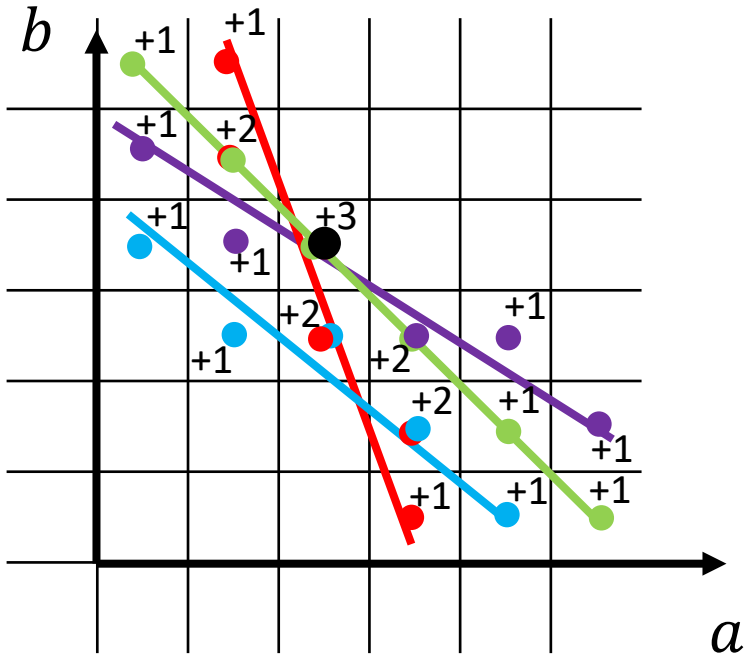




Hough line detection – Version 2

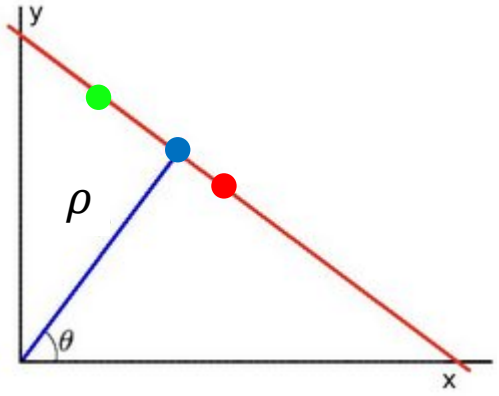
- 1. Quantize the parameter space “a-b” by dividing it into cells
- 2. For each point (x_1, y_1) on the detected edge in the “x-y” space, draw a line in the “a-b” space
- 3. Increase the value of a cell if a point (a, b) belongs to this cell
- 4. Cells receiving more than a certain number of votes are assumed to correspond to lines in the “x-y” space

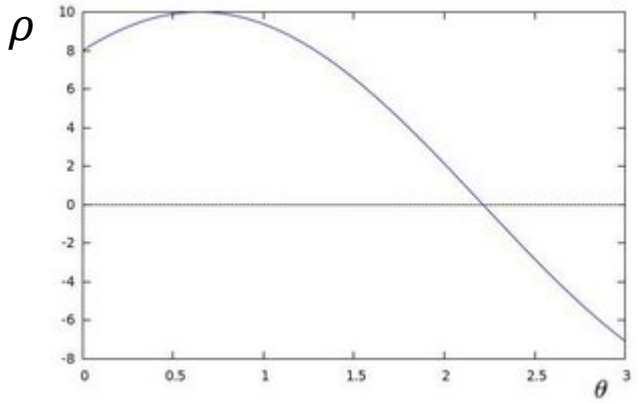


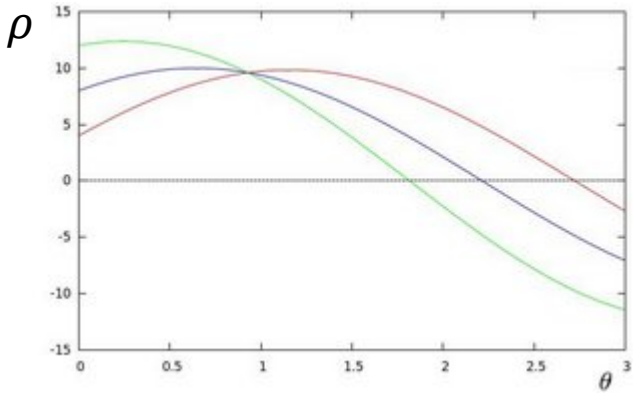


Hough line transform using Polar Coordinates

- It is better to use Polar Coordinate System than Cartesian Coordinate System to avoid singularity of vertical lines
- Parameters: (ρ_i, θ_i) $\rho = x_i \cos(\theta) + y_i \sin(\theta)$
- A point in the "x-y" space corresponds to a curve in the "ρ- θ" space
- An intersection of many curves in the "ρ- θ" space will define a line in the "x-y" space





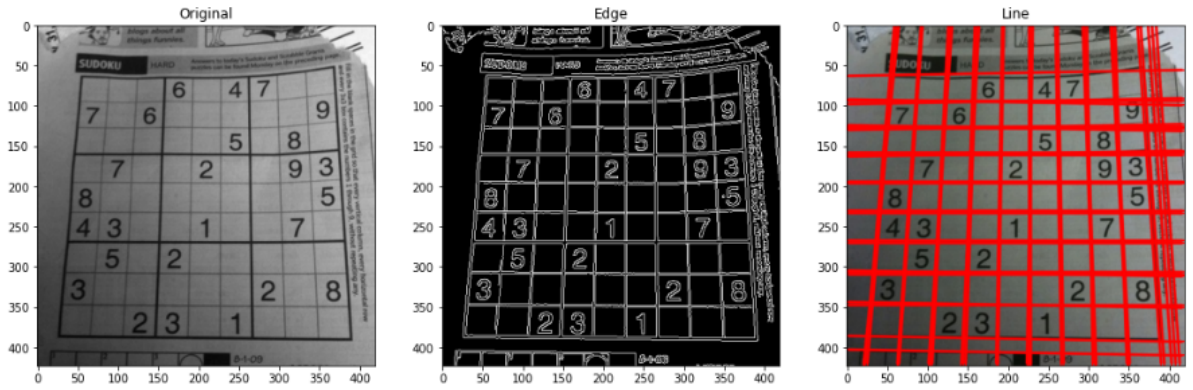


https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html

OpenCV examples – Hough line transform

```python
#import required libraries
import cv2 # OpenCV library
import numpy as np # Numpy library for scientific computing
import matplotlib.pyplot as plt # Matplotlib library for plotting

#canny edge detection
img = cv2.imread('sudoku.jpg',cv2.IMREAD_GRAYSCALE)
img_bgr = cv2.imread('sudoku.jpg')
edges = cv2.Canny(img, threshold1 = 50, threshold2 = 100, apertureSize = 3)

#hough transformation
lines = cv2.HoughLines(edges, rho = 1, theta = np.pi/180, threshold = 160)

#draw lines
l,_,_ = lines.shape
for i in range(l):
    rho = lines[i][0][0]
    theta = lines[i][0][1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img_bgr, (x1,y1), (x2,y2), (0,0,255), 2, cv2.LINE_AA)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

#plot the image
fig, (ax1, ax2, ax3) = plt.subplots(figsize = (18, 10), ncols = 3)
ax1.imshow(img, cmap='gray'), ax1.set_title("Original")
ax2.imshow(edges, cmap='gray'), ax2.set_title("Edge")
ax3.imshow(img_rgb, cmap='gray'), ax3.set_title("Line")
```

```
(<matplotlib.image.AxesImage at 0x1b396854d08>, Text(0.5, 1.0, 'Line'))
```



Hough line detection - Summary

Advantages

- Conceptually simple
- Easy implementation
- Handles missing and occluded data very gracefully
- Can be adapted to many types of forms, not just linesn

Disadvantages

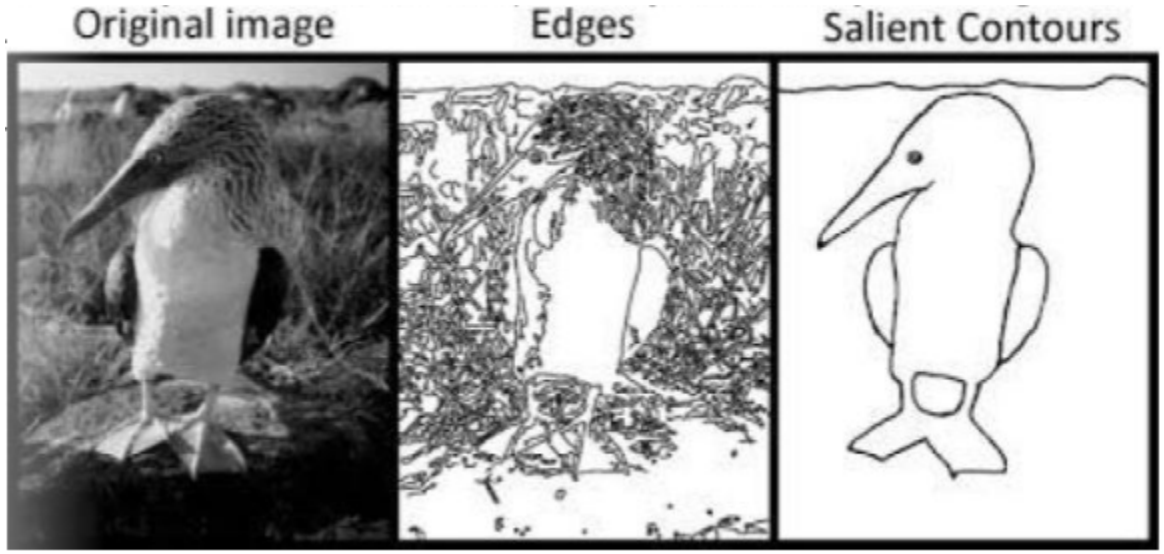
- Computationally complex for objects with many parameters
- Looks for only one single type of object
- The length and the position of a line segment cannot be determined
- Co-linear line segments cannot be separated

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

Contour Detection

Contour detection

- Contour:
  - A curve joining all the continuous points (along the boundary), having same color or intensity.
  - Useful for shape analysis and object detection and recognition.
  - For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.



https://medium.com/swlh/contours-in-images-a58b4c12c0ff

OpenCV implementation

- Following a method proposed by Suzuki and Abe in 1985
- Key idea: trace borders by connectivity and determine topological hierarchy

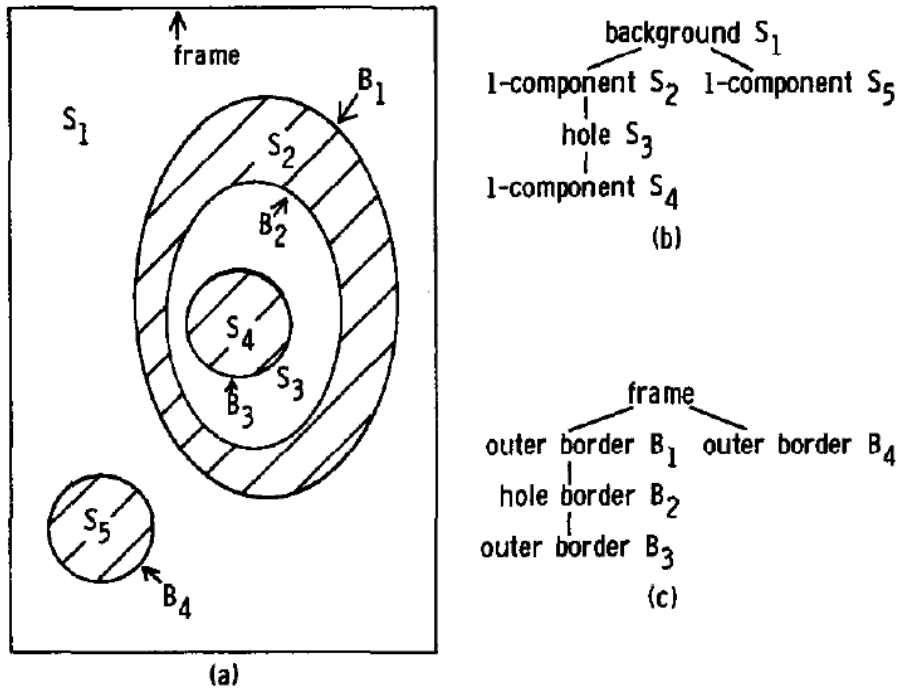


FIG. 1. Surroundness among connected components (b) and among borders (c).

Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)

Contour characteristics

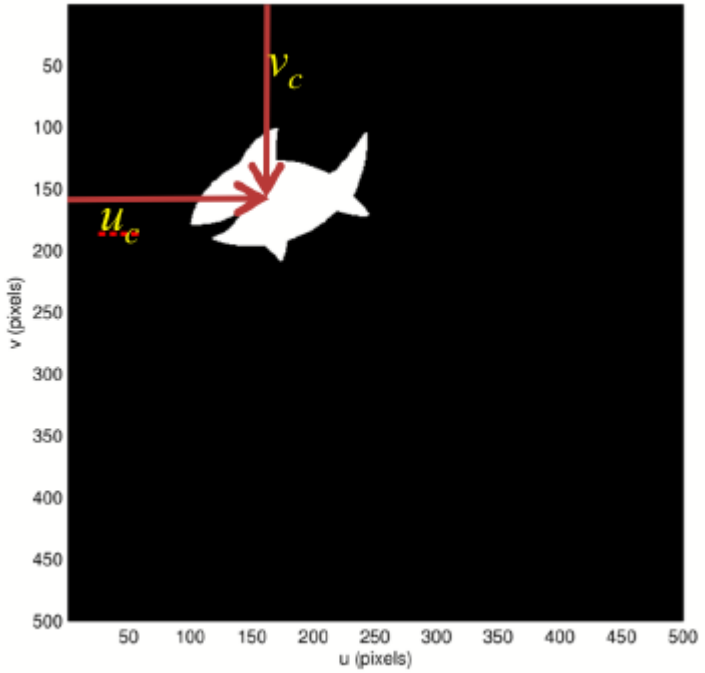
- Moment (similar to moment of mass in physics)

$$m_{pq} = \sum_{(u,v)\in \mathbf{I}} u^p v^q \mathbf{I}[u,v]$$

- Area (m_{00} for binary image)

$$m_{00} = \sum_{(u,v)\in \mathbf{I}} \mathbf{I}[u,v]$$

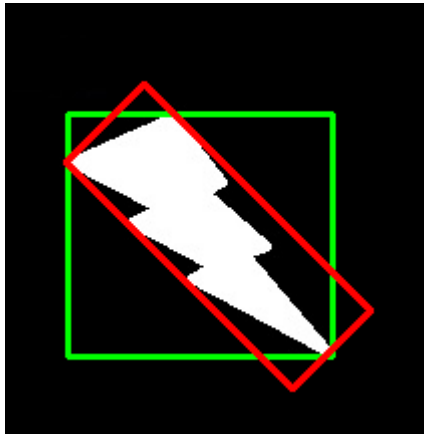
- Centroid $u_c = \frac{m_{10}}{m_{00}}, \quad v_c = \frac{m_{01}}{m_{00}}$



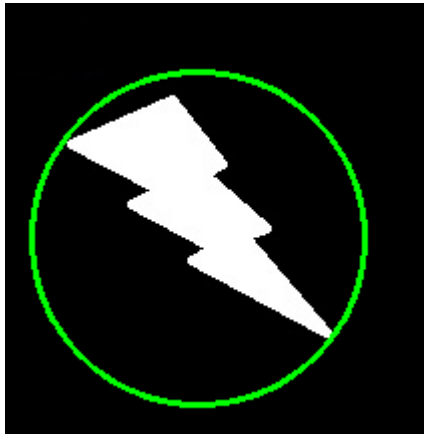
Contour characteristics

- Some other characteristics

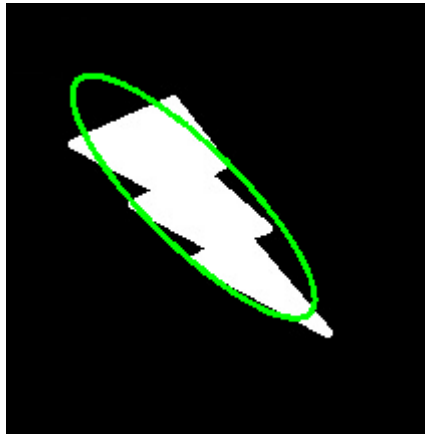
Bounding rectangle



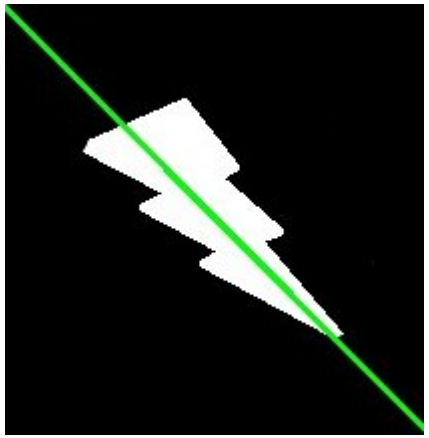
Minimum enclosing circle



Fitting an ellipse



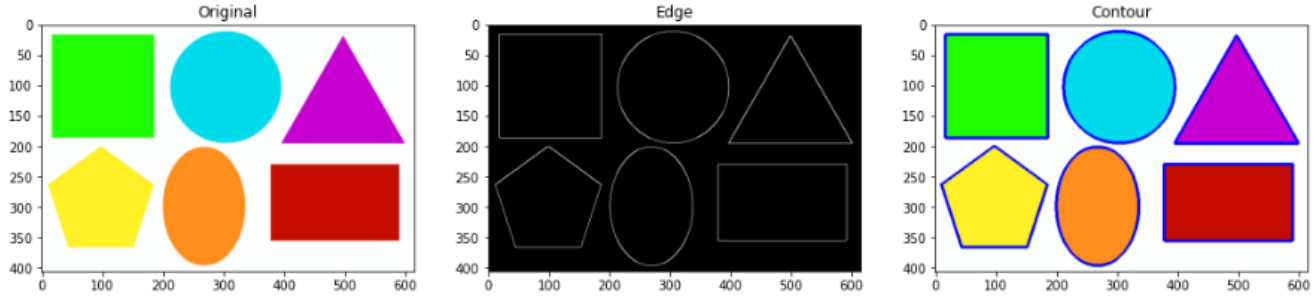
Fitting a line



OpenCV examples – Contour detection

```python
#import required 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

#canny edge detection
img_bgr = cv2.imread('shapes.png')
img_gray = cv2.imread('shapes.png',cv2.IMREAD_GRAYSCALE)
img_rgb_origin = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
edges = cv2.Canny(img_gray, threshold1 = 50, threshold2 = 100, apertureSize = 3)

#contour detection
img2, contours, hierarchy = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(img_bgr, contours, -1, (255, 0, 0), 3)
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)

#plot the image
fig, (ax1, ax2, ax3) = plt.subplots(figsize = (18, 10), ncols = 3)
ax1.imshow(img_rgb_origin), ax1.set_title("Original")
ax2.imshow(edges, cmap='gray'), ax2.set_title("Edge")
ax3.imshow(img_rgb), ax3.set_title("Contour")

#calculate moments and areas
for i, cnt in enumerate(contours):
    m = cv2.moments(cnt)
    area = cv2.contourArea(cnt)
    print("shape", i, ": m00 -", m['m00'], " area -", area)
```

```
shape 0 : m00 - 26709.0  area - 26709.0
shape 1 : m00 - 20900.5  area - 20900.5
shape 2 : m00 - 20065.0  area - 20065.0
shape 3 : m00 - 18234.5  area - 18234.5
shape 4 : m00 - 28729.0  area - 28729.0
shape 5 : m00 - 26514.0  area - 26514.0
```



Corner Detection

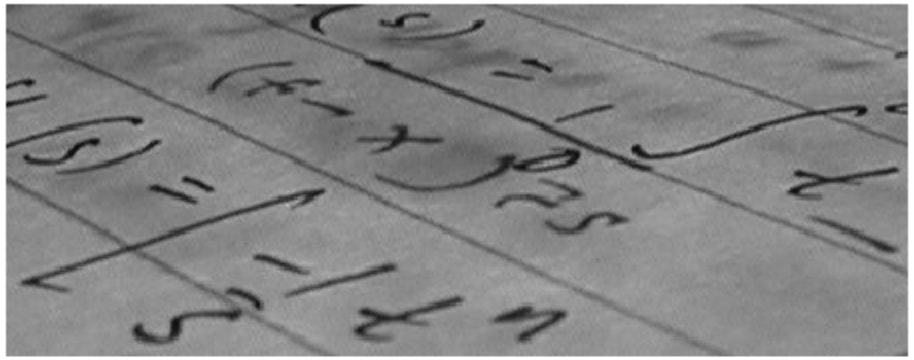
Why corner detection?

- Motivating problem:
  - Can you find the patches below in the right image?

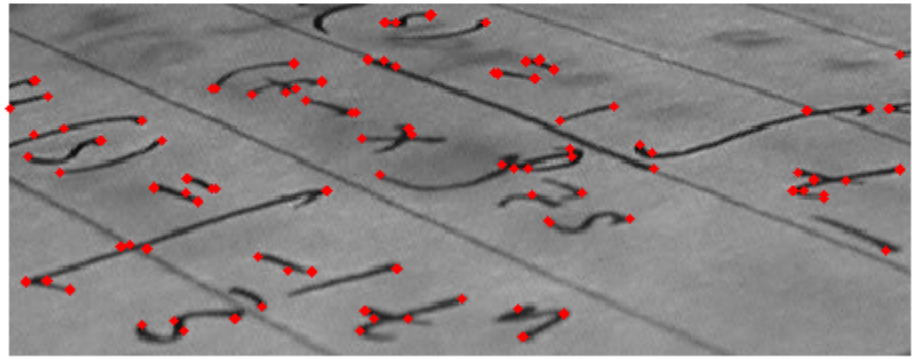
https://medium.com/analytics-vidhya/corner-detection-using-opencv-13998a679f76

Corner detection

- Corners are usually more unique and robust features



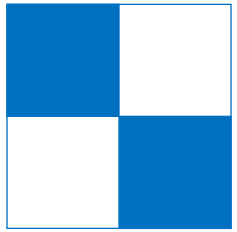
- Corner detection is an approach to extract certain kinds of features and infer the contents of an image



- Frequently used in motion detection, image registration, video tracking, image mosaicking, panorama stitching, 3D reconstruction and object recognition

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

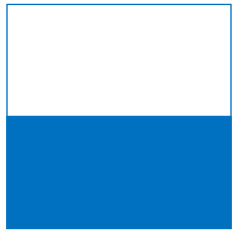
Corners vs edges



Intensity change in x → Large

Intensity change in y → Large

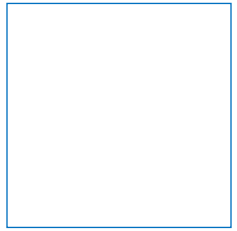
Corner



Intensity change in x → Small

Intensity change in y → Large

Edge



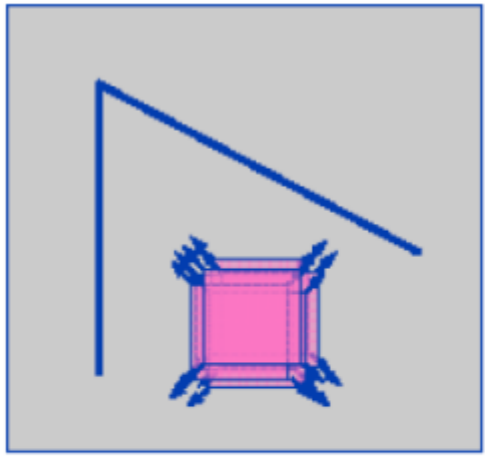
Intensity change in x → Small

Intensity change in y → Small

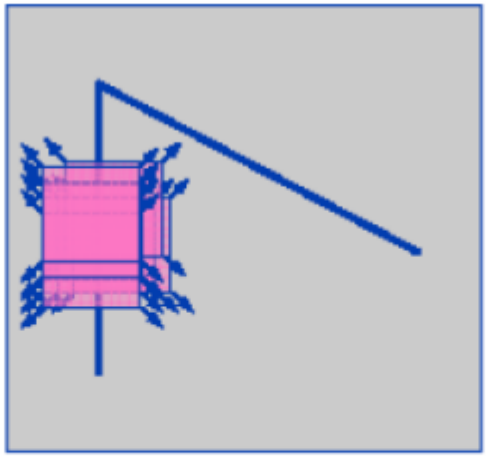
Nothing

Harris corner detector

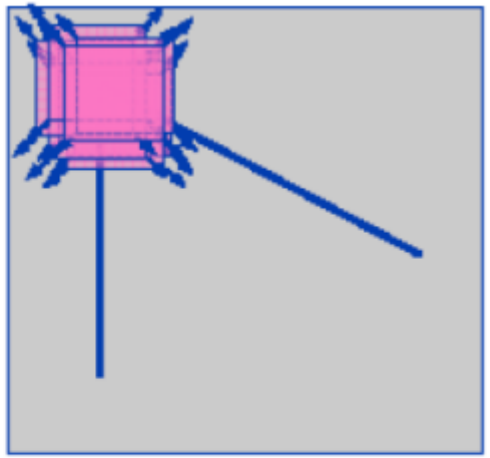
- Proposed by Chris Harris and Mike Stephens in 1988
- Shift patch by [u,v] and compute change in intensity



Flat region: no intensity change in all directions



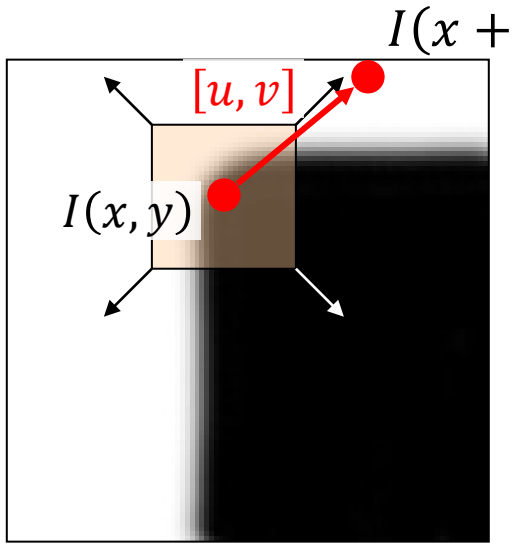
Edge: no change along edge directions



Corner: significant intensity change in many directions

Measure change as intensity difference:

$$I(x+u, y+v) - I(x, y)$$

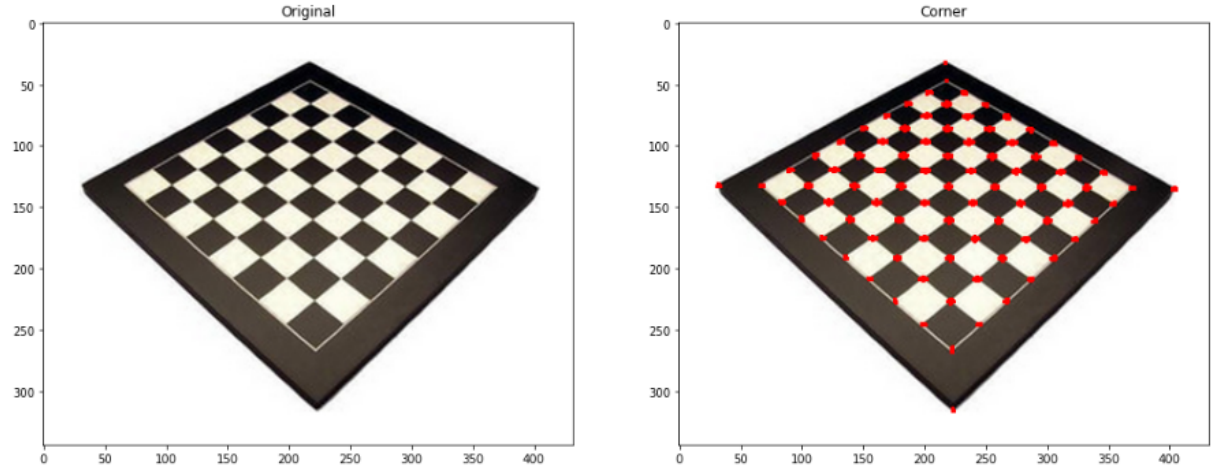


"corner":
significant change in all directions

OpenCV examples – Corner detection

```python
#import required 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

# image is loaded with imread command
img_bgr = cv2.imread('chessboard.jpg')
img_rgb_origin = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)

# modify the data type, setting to 32-bit floating point
img_gray = np.float32(img_gray)

# apply the cv2.cornerHarris method to detect the corners with appropriate values as input parameters
corners = cv2.cornerHarris(img_gray, blockSize = 2, ksize = 3, k = 0.03)

# Results are marked through the dilated corners
corners = cv2.dilate(corners, None)

# Reverting back to the original image, with optimal threshold value
img_bgr[corners > 0.01 * corners.max()] = [0, 0, 255]
img_rgb = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)

#plot the image
fig, (ax1, ax2) = plt.subplots(figsize = (18, 10), ncols = 2)
ax1.imshow(img_rgb_origin), ax1.set_title("Original")
ax2.imshow(img_rgb), ax2.set_title("Corner")
```

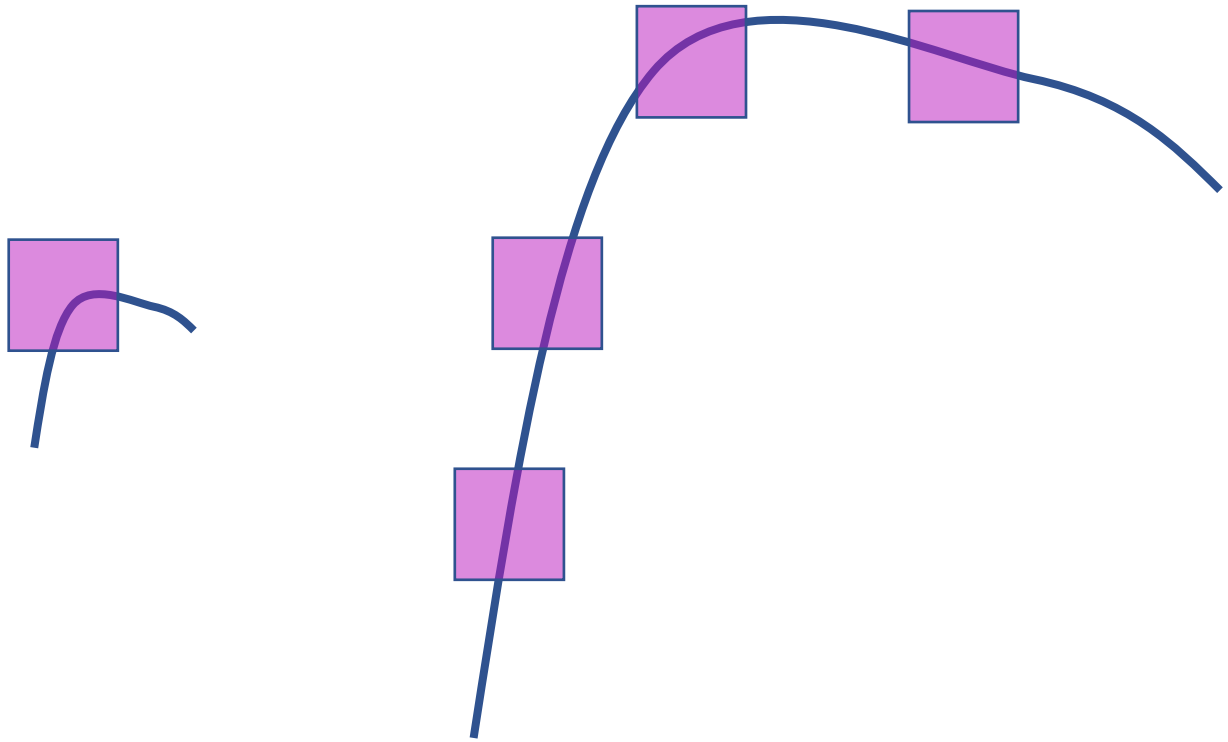
```
(<matplotlib.image.AxesImage at 0x1b39c1f73c8>, Text(0.5, 1.0, 'Corner'))
```



Problems with Harris corner detector

- But what about scale?

- It is rotation invariant



SIFT, SURF, and ORB

SIFT

- Scale-Invariant Feature Transform
- Developed by D. Lowe in 2004
- Four main steps
  - 1. Scale-space Extrema Detection
  - 2. Keypoint Localization
  - 3. Orientation Assignment
  - 4. Keypoint Descriptor

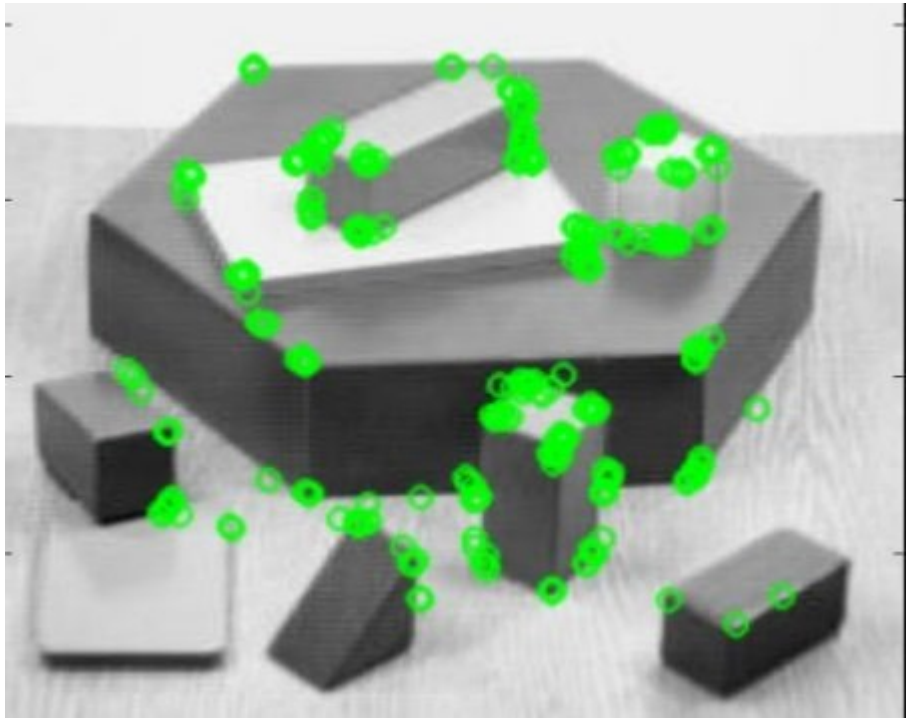
D.Lowe. Distinctive Image Features from Scale-Invariant Keypoints. University of British Columbia.

SURF

- Speeded-Up Robust Features
- Developed by H. Bay, T. Tuytelaars, and L. Van Gool in 2006.
- A speeded-up version of SIFT

ORB

- Oriented FAST and Rotated BRIEF
- Developed by Ethan Rublee et al. in 2011.
- Aims to provide a fast and efficient alternative to SIFT

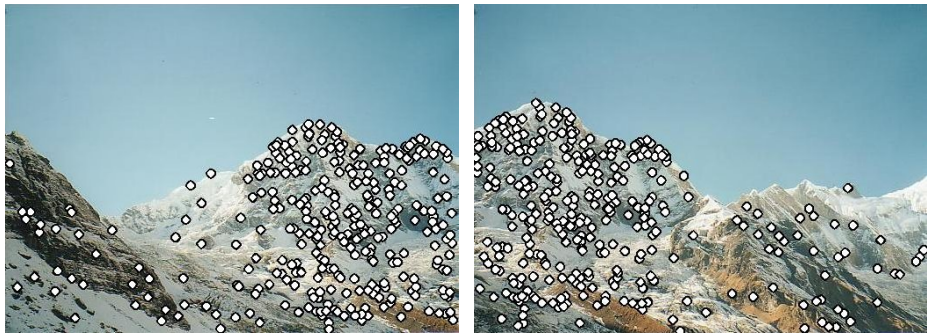


https://en.wikipedia.org/wiki/Oriented_FAST_and_rotated_BRIEF

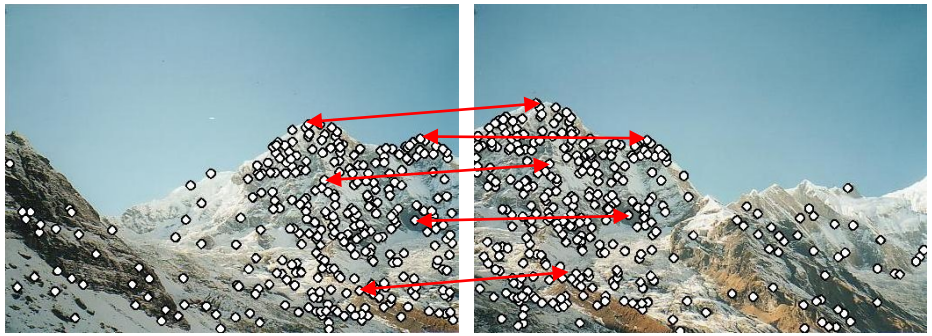
Example application: Image stitching

Example application: Image stitching

- Feature detection



- Matching

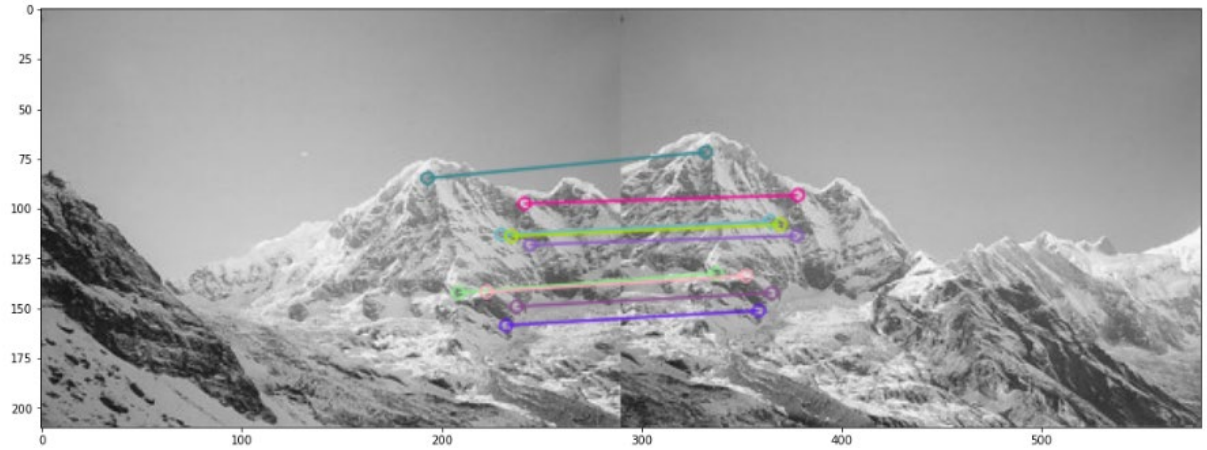


- Stitching

OpenCV examples - SIFT

```python
#import required 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

img1 = cv2.imread('mountain1.jpg',cv2.IMREAD_GRAYSCALE)          # queryImage
img2 = cv2.imread('mountain2.jpg',cv2.IMREAD_GRAYSCALE)          # trainImage

# Initiate SIFT detector
sift  = cv2.xfeatures2d.SIFT_create()

# find the keypoints and descriptors with SIFT
kp1, des1 = sift.detectAndCompute(img1, None)
kp2, des2 = sift.detectAndCompute(img2, None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)

plt.figure(figsize = (18, 10))
plt.imshow(img3)
```

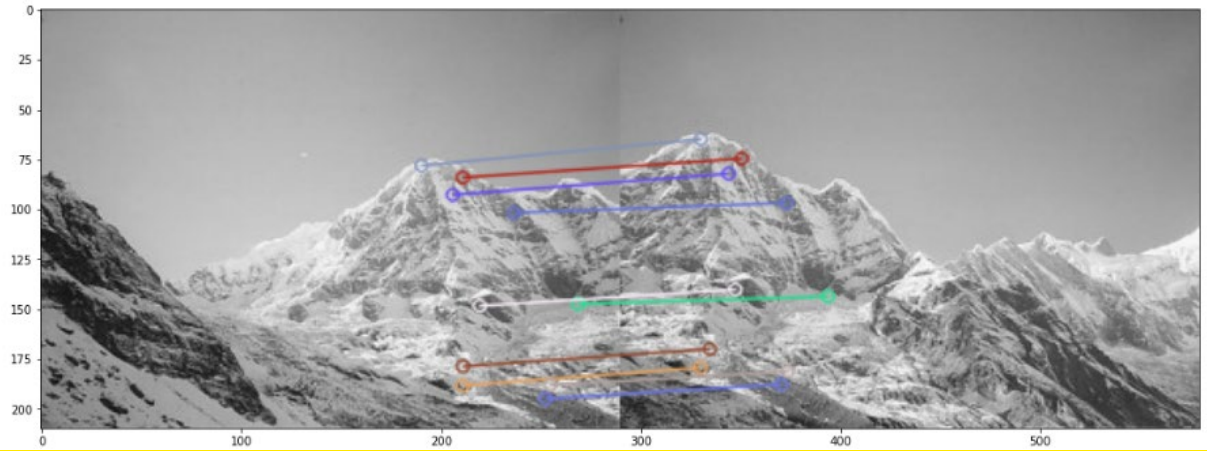
```
<matplotlib.image.AxesImage at 0x1b38d5c5348>
```



OpenCV examples - SURF

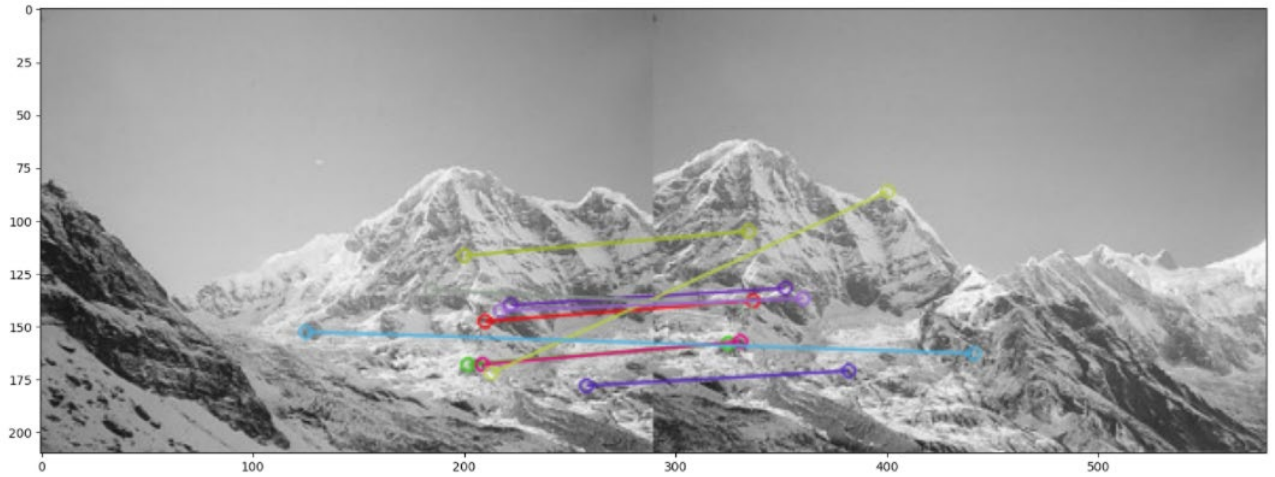
```python
#import required libraries
import cv2 # OpenCV library
import numpy as np # Numpy library for scientific computing
import matplotlib.pyplot as plt # Matplotlib library for plotting
# show plots in notebook
%matplotlib inline

img1 = cv2.imread('mountain1.jpg',cv2.IMREAD_GRAYSCALE)          # queryImage
img2 = cv2.imread('mountain2.jpg',cv2.IMREAD_GRAYSCALE)          # trainImage

# Initiate SURF detector
surf  = cv2.xfeatures2d.SURF_create()

# find the keypoints and descriptors with SURF
kp1, des1 = surf.detectAndCompute(img1, None)
kp2, des2 = surf.detectAndCompute(img2, None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)

plt.figure(figsize = (18, 10))
plt.imshow(img3)
```

```
<matplotlib.image.AxesImage at 0x1b38d62e908>
```



OpenCV examples - ORB

```python
#import required libraries
import cv2 # OpenCV library
import numpy as np # Numpy library for scientific computing
import matplotlib.pyplot as plt # Matplotlib library for plotting

img1 = cv2.imread('../mountain1.jpg',cv2.IMREAD_GRAYSCALE)          # queryImage
img2 = cv2.imread('../mountain2.jpg',cv2.IMREAD_GRAYSCALE)          # trainImage

# Initiate ORB detector
orb  = cv2.ORB_create()

# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1, None)
kp2, des2 = orb.detectAndCompute(img2, None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)

# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img1, kp1, img2, kp2, matches[:10], None, flags=2)

plt.figure(figsize = (18, 10))
plt.imshow(img3)
plt.show()
```



Marker Detection

What if an image does not have unique corners?

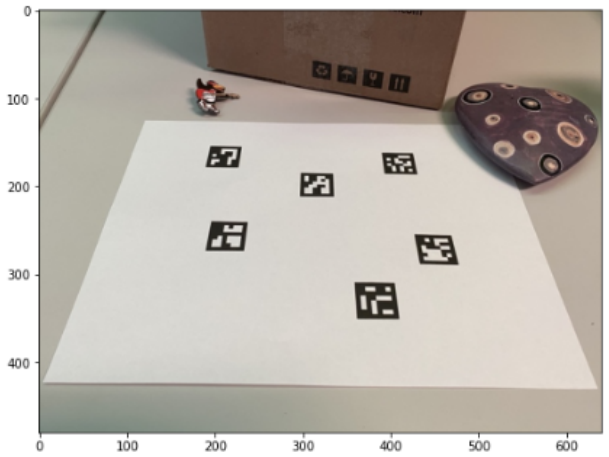
- Fiducial markers can be used for object detection
- A popular one is the ArUco (Augmented Reality University of Cordoba) marker proposed by S. Garrido-Jurado et al. in 2014
- Popular in AR, robotics, etc.

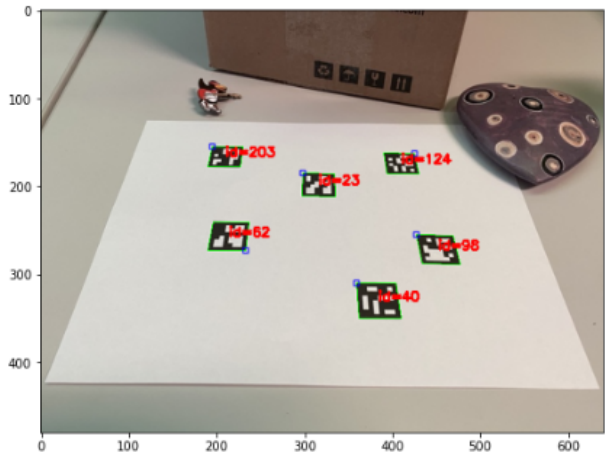
S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. 2014. "Automatic generation and detection of highly reliable fiducial markers under occlusion". Pattern Recogn. 47, 6 (June 2014), 2280-2292. DOI=10.1016/j.patcog.2014.01.005
https://youtu.be/nsu9tNIJ6F0

OpenCV examples – ArUco marker detection

```python
#import required 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

# Detect ArUco markers
# Image is loaded with imread command
img_bgr = cv2.imread('aruco.jpg')
img_rgb_origin = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)
img_rgb = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2RGB)

# Load the predefined dictionary
dictionary = cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)

# Initialize the detector parameters using default values
parameters =  cv2.aruco.DetectorParameters_create()

# Detect the markers in the image
markerCorners, markerIds, rejectedCandidates = cv2.aruco.detectMarkers(img_rgb, dictionary, parameters=parameters)

cv2.aruco.drawDetectedMarkers(img_rgb, markerCorners, markerIds)

#plot the image
fig, (ax1, ax2) = plt.subplots(figsize = (18, 10), ncols = 2)
ax1.imshow(img_rgb_origin)
ax2.imshow(img_rgb)
```

```
<matplotlib.image.AxesImage at 0x1b38ec34dc8>
```





Face Detection

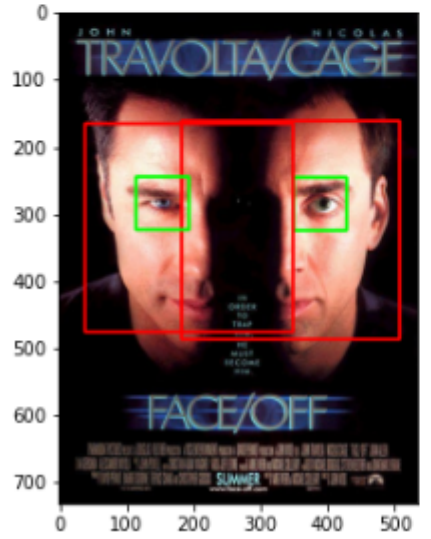
Face detection

- The methods introduced above are all based on geometry
- There are also methods using machine learning
- One example is face detection using Haar Cascades
  - A cascade function is trained from a lot of positive and negative images
- OpenCV comes with a trainer as well as a detector
- Pre-trained classifiers are available for direct use

OpenCV examples – Face detection

```python
#import required 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

#load the classifiers downloaded
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')

#read the image and convert to grayscale format
img = cv2.imread('face_off_1.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#calculate coordinates
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),3)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    #draw bounding boxes around detected features
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),3)
face_1 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#read the image and convert to grayscale format
img = cv2.imread('face_off_2.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#calculate coordinates
faces = face_cascade.detectMultiScale(gray, 1.1, 4)
for (x,y,w,h) in faces:
    cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),3)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = img[y:y+h, x:x+w]
    eyes = eye_cascade.detectMultiScale(roi_gray)
    #draw bounding boxes around detected features
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),3)
face_2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#plot the image
fig, (ax1, ax2) = plt.subplots(figsize = (9, 5), ncols = 2)
ax1.imshow(face_1)
ax2.imshow(face_2)
```

```
<matplotlib.image.AxesImage at 0x1b39c13d048>
```



Object Tracking

Object tracking vs object detection

- Option 1
  - Detect objects on each frame of the input video
- Option 2
  - Detect objects once and have the objects tracked every subsequent frame
  - Faster and more efficient

https://www.youtube.com/watch?v=JFVAy_uC0V4

OpenCV provides 8 object tracking methods

1. **BOOSTING Tracker:** Based on the same algorithm used to power the machine learning behind Haar cascades (AdaBoost), but like Haar cascades, is over a decade old. This tracker is slow and doesn't work very well. Interesting only for legacy reasons and comparing other algorithms. *(minimum OpenCV 3.0.0)*
2. **MIL Tracker:** Better accuracy than BOOSTING tracker but does a poor job of reporting failure. *(minimum OpenCV 3.0.0)*
3. **KCF Tracker:** Kernelized Correlation Filters. Faster than BOOSTING and MIL. Similar to MIL and KCF, does not handle full occlusion well. *(minimum OpenCV 3.1.0)*
4. **CSRT Tracker:** Discriminative Correlation Filter (with Channel and Spatial Reliability). Tends to be more accurate than KCF but slightly slower. *(minimum OpenCV 3.4.2)*
5. **MedianFlow Tracker:** Does a nice job reporting failures; however, if there is too large of a jump in motion, such as fast moving objects, or objects that change quickly in their appearance, the model will fail. *(minimum OpenCV 3.0.0)*
6. **TLD Tracker:** I'm not sure if there is a problem with the OpenCV implementation of the TLD tracker or the actual algorithm itself, but the TLD tracker was incredibly prone to false-positives. I do not recommend using this OpenCV object tracker. *(minimum OpenCV 3.0.0)*
7. **MOSSE Tracker:** Very, *very* fast. Not as accurate as CSRT or KCF but a good choice if you need pure speed. *(minimum OpenCV 3.4.1)*
8. **GOTURN Tracker:** The only deep learning-based object detector included in OpenCV. It requires additional model files to run (will not be covered in this post). My initial experiments showed it was a bit of a pain to use even though it reportedly handles viewing changes well (my initial experiments didn't confirm this though). I'll try to cover it in a future post, but in the meantime, take a look at Satya's writeup. *(minimum OpenCV 3.2.0)*

OpenCV examples – Object tracking

```python
import cv2
import sys

(major_ver, minor_ver, subminor_ver) = (cv2.__version__).split('.')

if __name__ == '__main__' :

    # Set up tracker.
    # Instead of MIL, you can also use

    tracker_types = ['BOOSTING', 'MIL','KCF', 'TLD', 'MEDIANFLOW', 'GOTURN', 'MOSSE', 'CSRT']
    tracker_type = tracker_types[2]

    if int(minor_ver) < 3:
        tracker = cv2.Tracker_create(tracker_type)
    else:
        if tracker_type == 'BOOSTING':
            tracker = cv2.TrackerBoosting_create()
        if tracker_type == 'MIL':
            tracker = cv2.TrackerMIL_create()
        if tracker_type == 'KCF':
            tracker = cv2.TrackerKCF_create()
        if tracker_type == 'TLD':
            tracker = cv2.TrackerTLD_create()
        if tracker_type == 'MEDIANFLOW':
            tracker = cv2.TrackerMedianFlow_create()
        if tracker_type == 'GOTURN':
            tracker = cv2.TrackerGOTURN_create()
        if tracker_type == 'MOSSE':
            tracker = cv2.TrackerMOSSE_create()
        if tracker_type == "CSRT":
            tracker = cv2.TrackerCSRT_create()

    # Read video
    video = cv2.VideoCapture("chaplin.mp4")

    # Exit if video not opened.
    if not video.isOpened():
        print("Could not open video")
        sys.exit()

    # Read first frame.
    ok, frame = video.read()
    if not ok:
        print('Cannot read video file')
        sys.exit()

    # Define an initial bounding box
    bbox = (287, 23, 86, 320)

    # Uncomment the line below to select a different bounding box
    bbox = cv2.selectROI(frame, False)

    # Initialize tracker with first frame and bounding box
    ok = tracker.init(frame, bbox)

    while True:
        # Read a new frame
        ok, frame = video.read()
        if not ok:
            break

        # Start timer
        timer = cv2.getTickCount()

        # Update tracker
        ok, bbox = tracker.update(frame)

        # Calculate Frames per second (FPS)
        fps = cv2.getTickFrequency() / (cv2.getTickCount() - timer);

        # Draw bounding box
        if ok:
            # Tracking success
            p1 = (int(bbox[0]), int(bbox[1]))
            p2 = (int(bbox[0] + bbox[2]), int(bbox[1] + bbox[3]))
            cv2.rectangle(frame, p1, p2, (255,0,0), 2, 1)
        else :
            # Tracking failure
            cv2.putText(frame, "Tracking failure detected", (100,80), cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)

        # Display tracker type on frame
        cv2.putText(frame, tracker_type + " Tracker", (100,20), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50),2);

        # Display FPS on frame
        cv2.putText(frame, "FPS : " + str(int(fps)), (100,50), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (50,170,50), 2);

        # Display result
        cv2.imshow("Tracking", frame)

        # Exit if ESC pressed
        k = cv2.waitKey(10) & 0xff
        if k == 27 : break

    cv2.destroyAllWindows()
```

Object detection and tracking to monitor social distancing

https://youtu.be/nDkVyAL6MdY

What we have learnt in these two weeks

- Introduction to computer vision
  - Connection with image processing and machine learning
  - Tools available
- Image processing
  - Pixels
  - Perspective transforms
  - Colour spaces
  - Thresholding
  - Morphological operations
- Feature detection
  - Edge detection
  - Hough transform
  - Contour detection
  - Corner detection
  - SIFT, SURF, and ORB
  - Marker detection
  - Face detection
- Object tracking
  - 8 object tracking algorithms

Beyond what we have learnt

- Challenges of computer vision

> *The goal of computer vision is to extract useful information from images. This has proved a surprisingly challenging task; it has occupied thousands of intelligent and creative minds over the last four decades, and despite this we are still far from being able to build a general-purpose "seeing machine."*

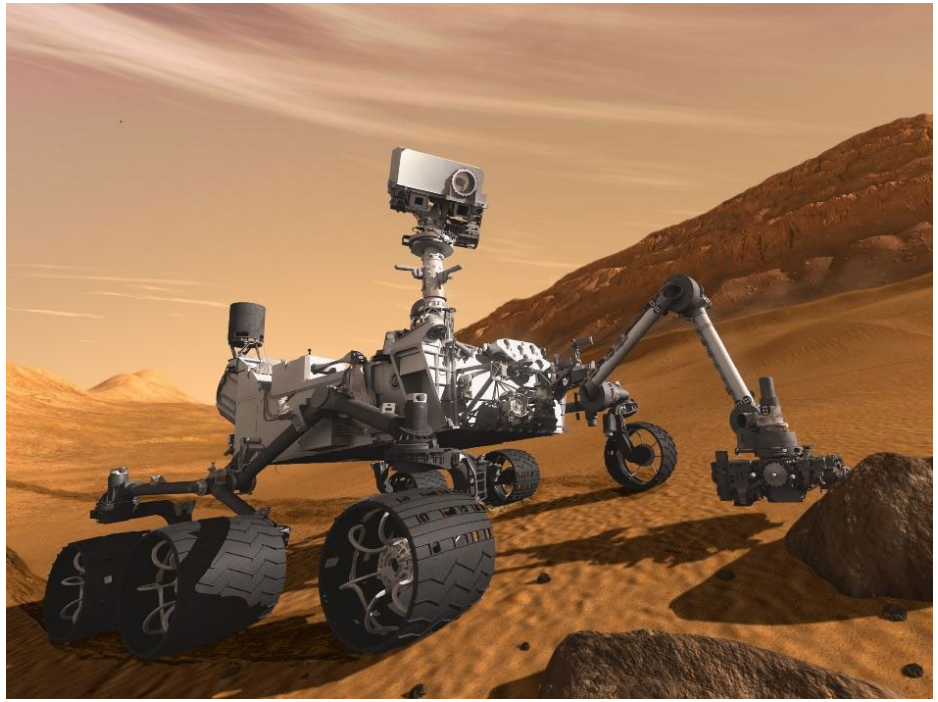
— Page 16, Computer Vision: Models, Learning, and Inference, 2012.

> *Making a computer see was something that leading experts in the field of Artificial Intelligence thought to be at the level of difficulty of a summer student's project back in the sixties. Forty years later the task is still unsolved and seems formidable.*

— Page xi, Multiple View Geometry in Computer Vision, 2004.

> *Perceptual psychologists have spent decades trying to understand how the visual system works and, even though they can devise optical illusions to tease apart some of its principles, a complete solution to this puzzle remains elusive*

— Page 3, Computer Vision: Algorithms and Applications, 2010.



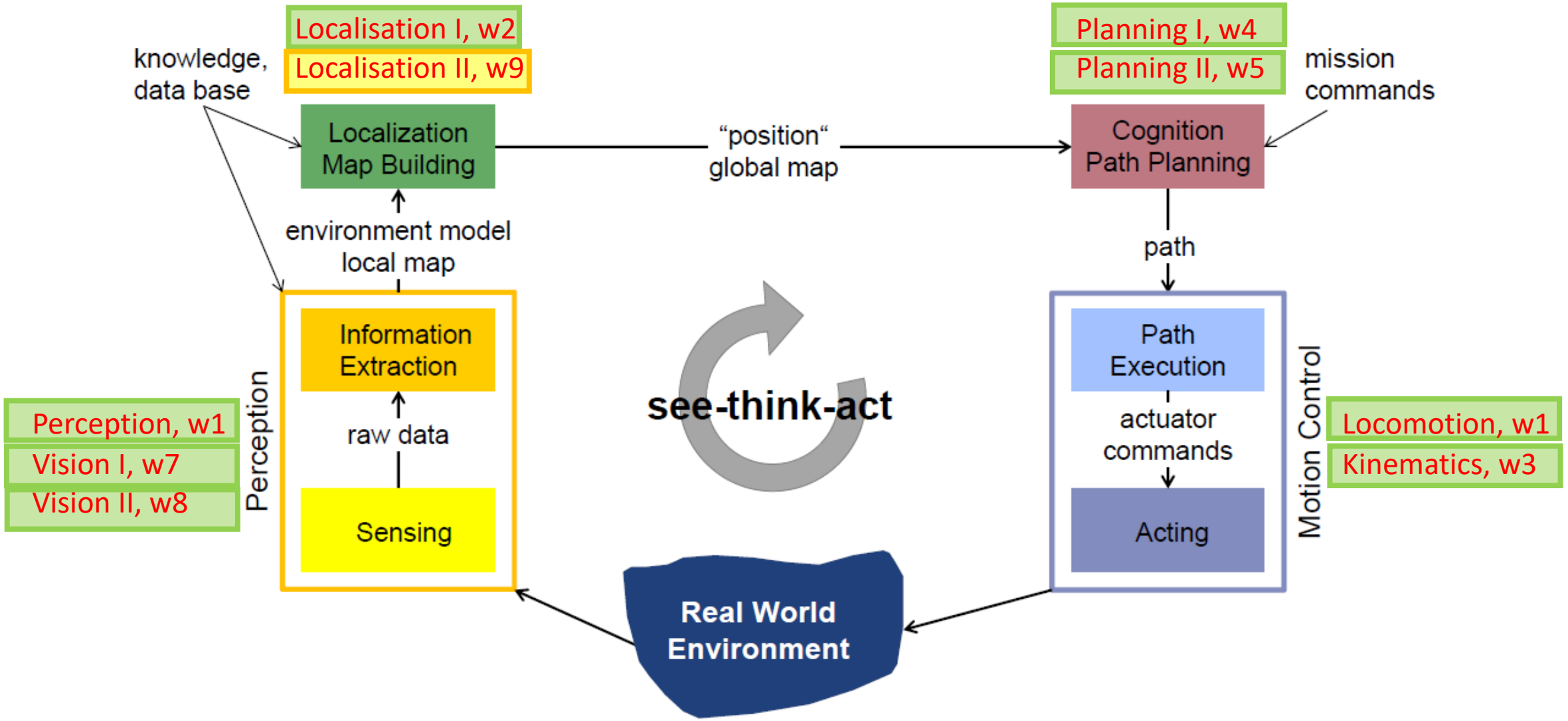
https://machinelearningmastery.com/what-is-computer-vision/
https://en.wikipedia.org/wiki/Computer_vision

Acknowledgment

- Many slides are adapted from Prof Emmanuel Agu
  - https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/

- And Juan Carlos Niebles and Ranjay Krishna
  - http://vision.stanford.edu/teaching/cs131_fall1920/slides/

Next week: Localisation II



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