Computer Vision

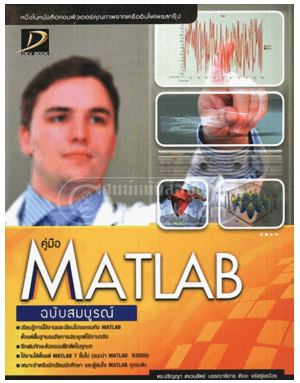
Parinya Sanguansat

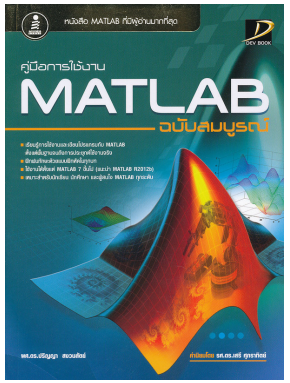
About me

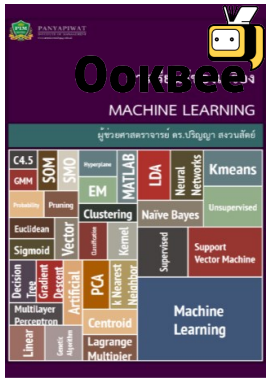
Lecturer

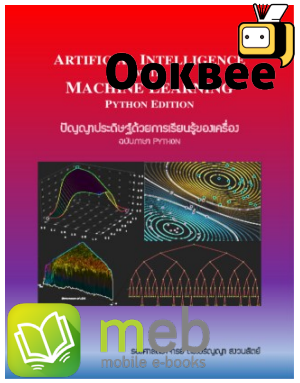
Assoc.Prof.Parinya Sanguansat, Ph.D.
รศ.ดร.ปริญญา สงวนสัตย์
Head of Computer Engineering and Artificial Intelligence,
Faculty of Engineering and Technology
Panyapiwat Institute of Management

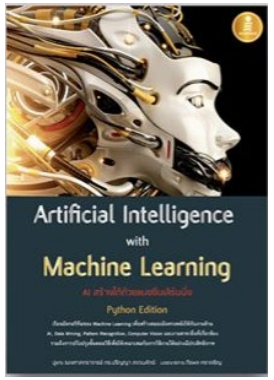
Writer



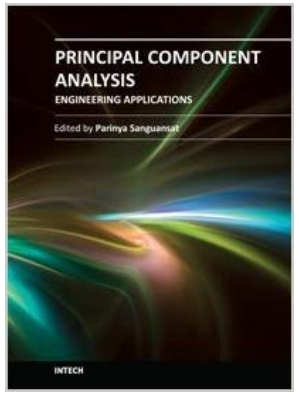


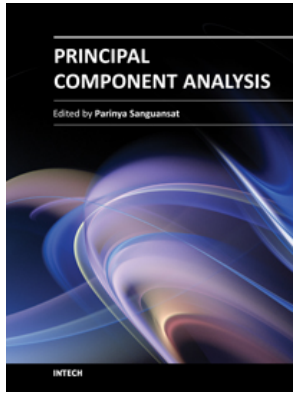






Editor





Youtuber

AI บ้าน บ้าน

Play around with image and video

- Create
- Visualize
- Import
- Export

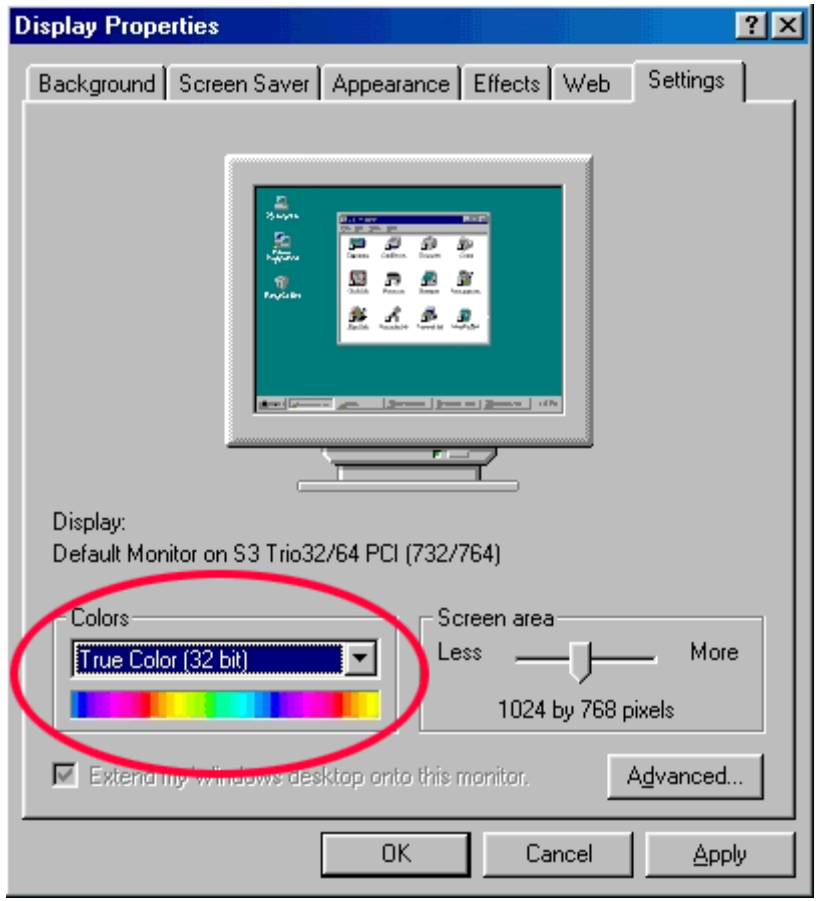
How to create an image

- Any data types can use but uint8, float32, and float64 are most frequently used

- For Python, numpy array is the medium

How to visualize an image

- You can create image in any data types but



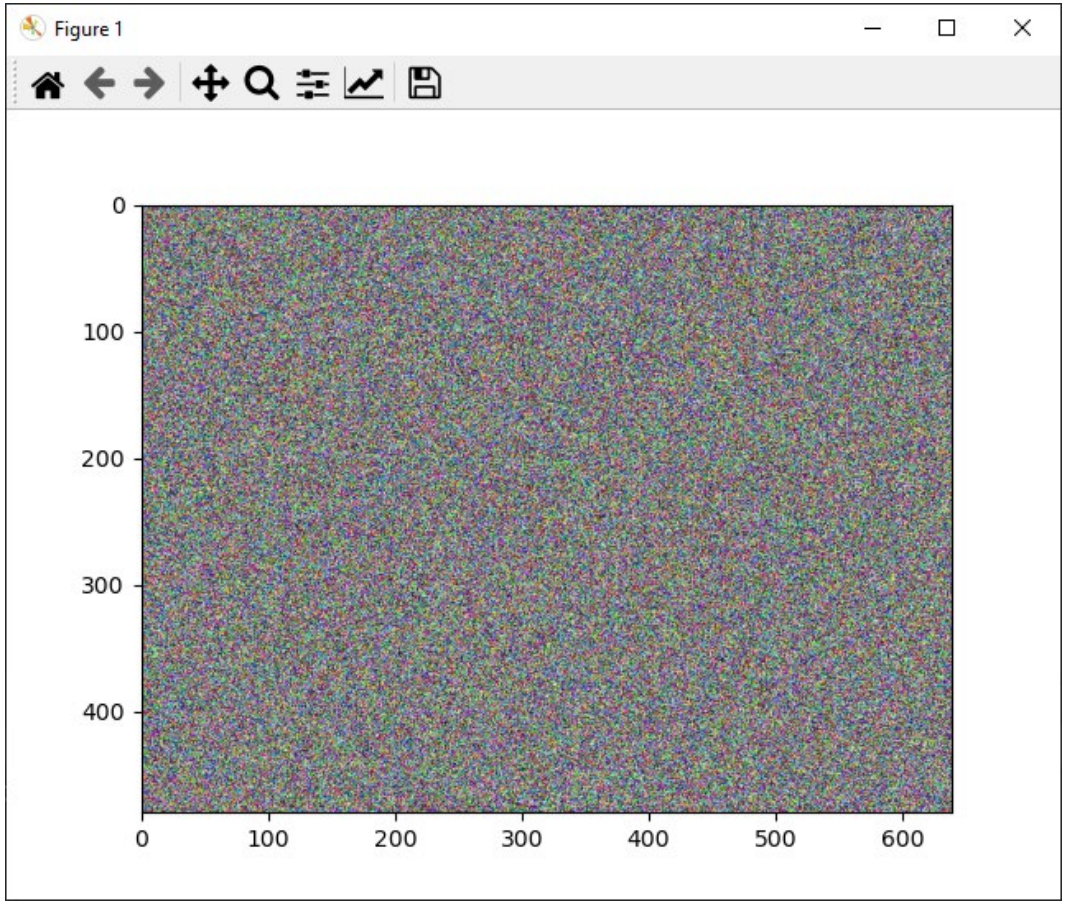
Ex: How to create an image

- Image of Noise in VGA resolution

```
import numpy as np
import matplotlib.pyplot as plt

im_shape = 480, 640, 3
im = np.random.rand(*im_shape)
plt.imshow(im)
plt.show()
```

Default data type is float64

What happens if we convert im to float32, float16, or uint8 and visualize with pyplot?

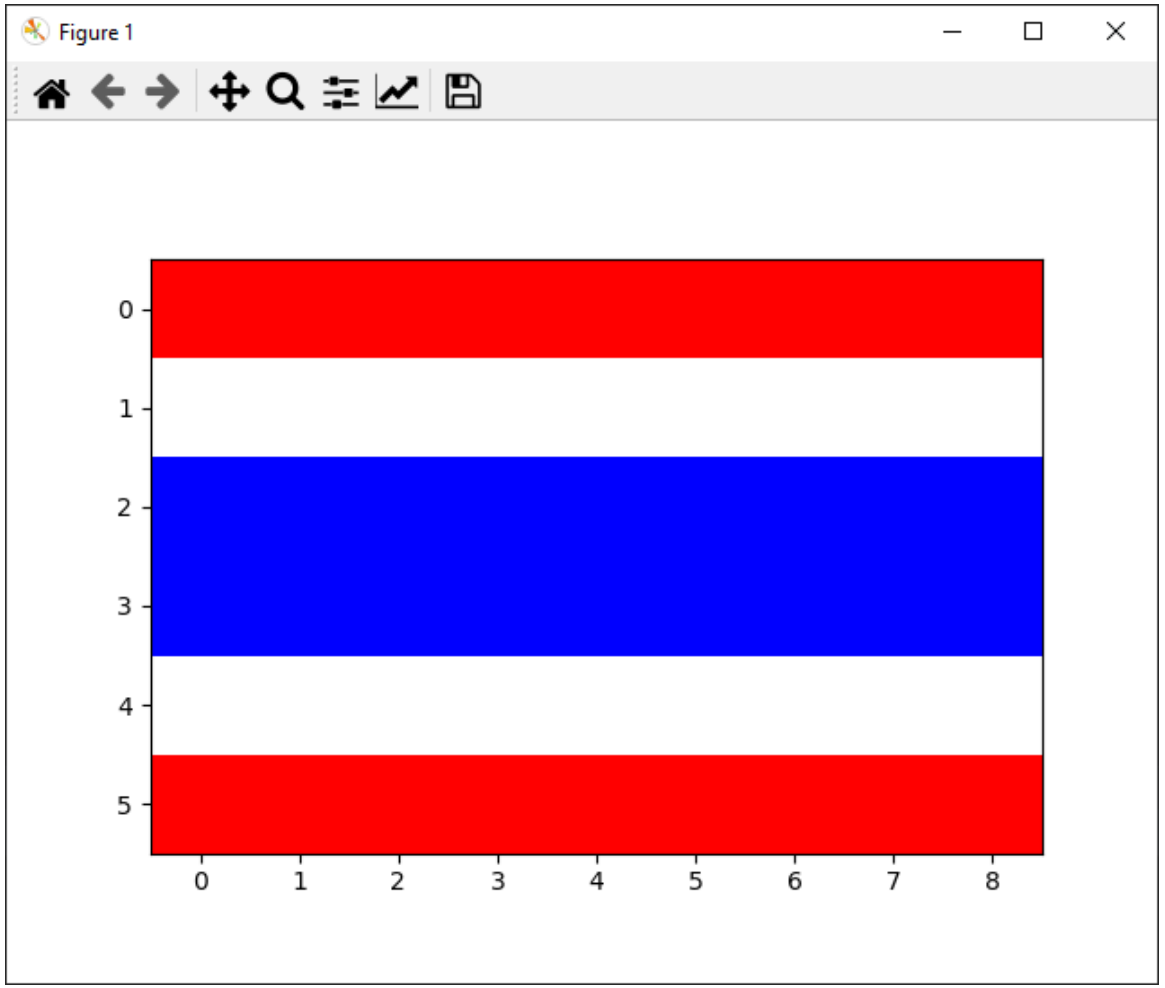


Ex: Flag of Thailand

```
Thai = np.ones((6, 9, 3))
Thai[[0, -1], :, 1:] = 0
Thai[2:4, :, :2] = 0
plt.imshow(Thai)
plt.show()
```

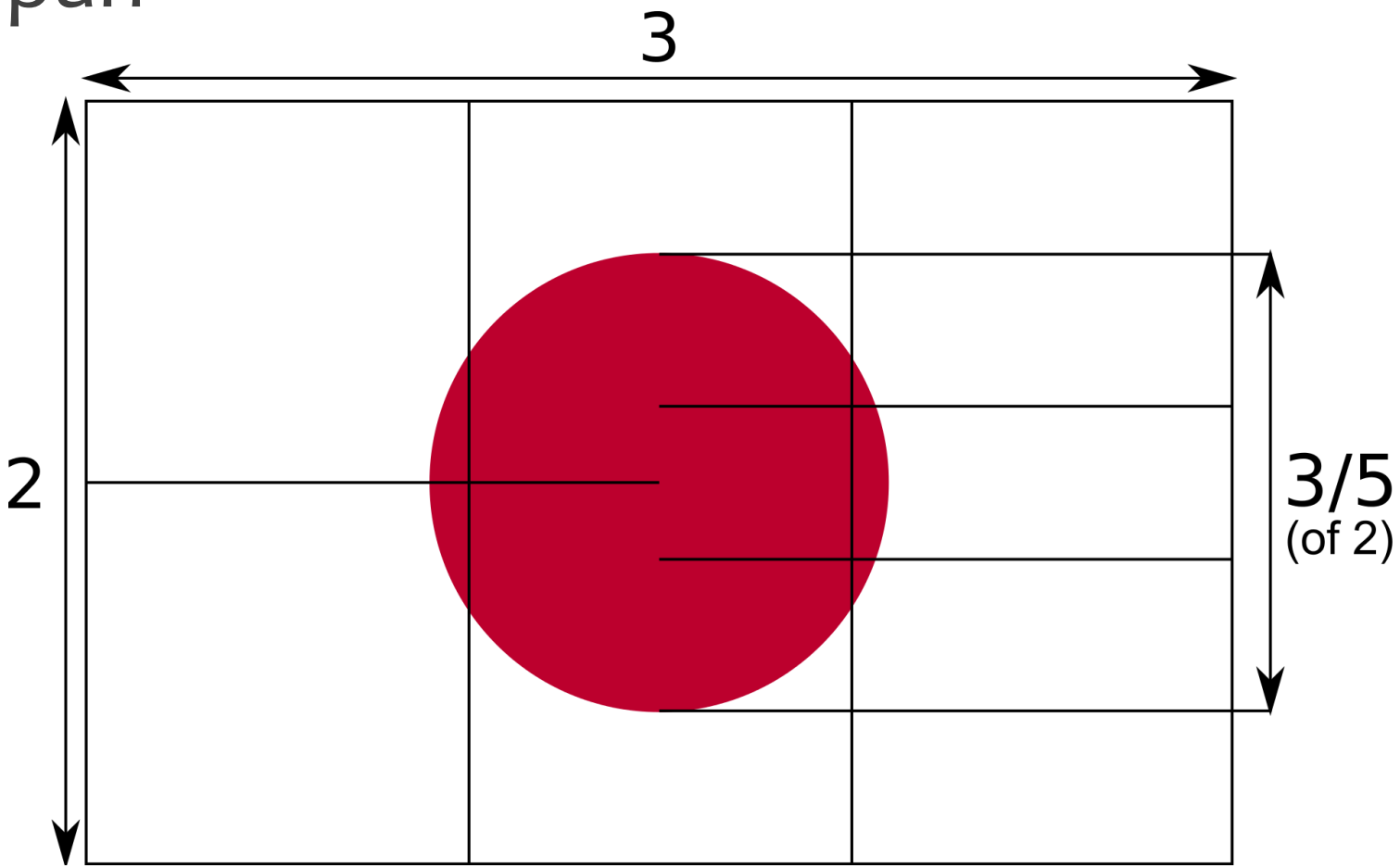
What happens if

- we initialize Thai with zeros
- we want to use unit8
- we use cv2.imshow() instead of plt.imshow()



5 min. challenge I

- Flag of Japan



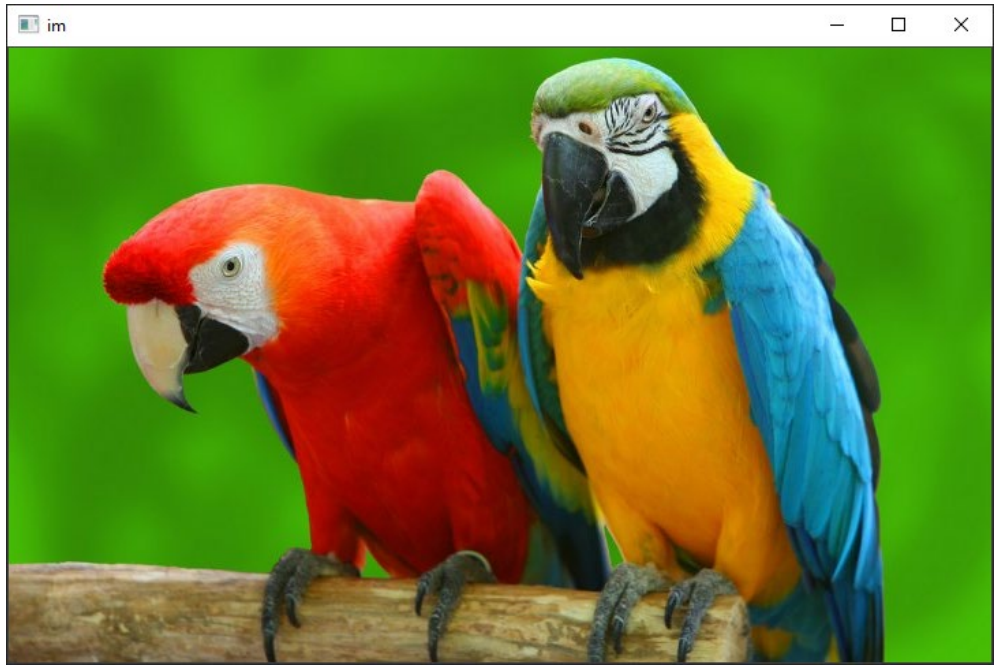
How to import an image

```
import matplotlib.pyplot as plt
im = plt.imread('parrot.jpg')
plt.imshow(im)
plt.show()
```

```
from PIL import Image
im = Image.open('parrot.jpg')
im = np.array(im)
plt.imshow(im)
plt.show()
```

```
import cv2
im = cv2.imread('parrot.jpg')
cv2.imshow('im', im)
cv2.waitKey()
```

Matplotlib can only read PNGs natively. Further image formats are supported via the optional dependency on Pillow



How to export an image

- Choose image file extension
  - Lossless: png, gif, tiff, bmp
  - Lossy: jpg
  - Multiple images: gif, tiff
  - Light weight: gif
  - High quality: tiff

```
cv2.imwrite('Thai.png', np.uint8(Thai[:, :, ::-1]*255))
```

```
im = Image.fromarray(np.uint8(Thai*255))
im.save('Thai.jpg')
```

```
im2 = Image.fromarray(np.uint8(255-Thai*255))
im.save('Thai.gif', save_all=True, append_images=[im2, im]*20)
```

1st frame

2nd frame

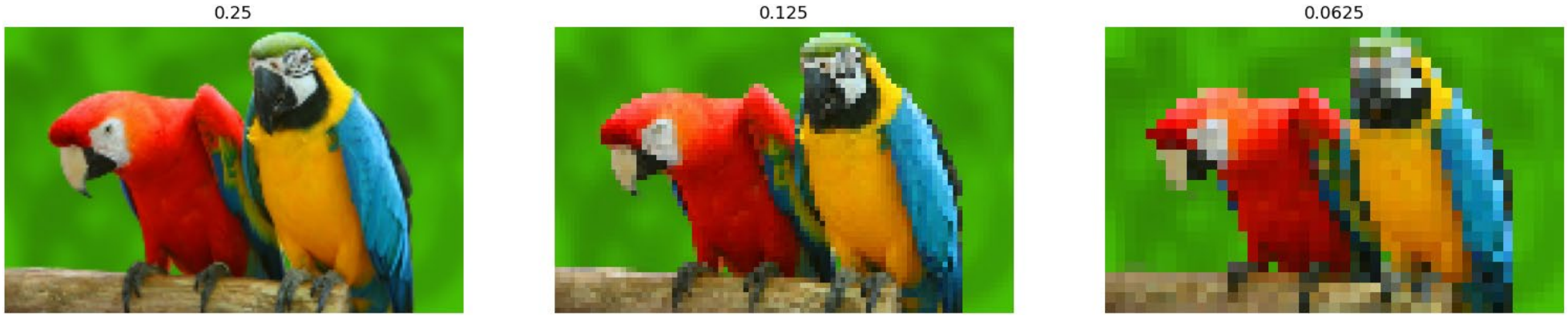
Remarks

- Can images have more than 8 bit per channel?
- Can images have more than 3 channel values?

Resolutions

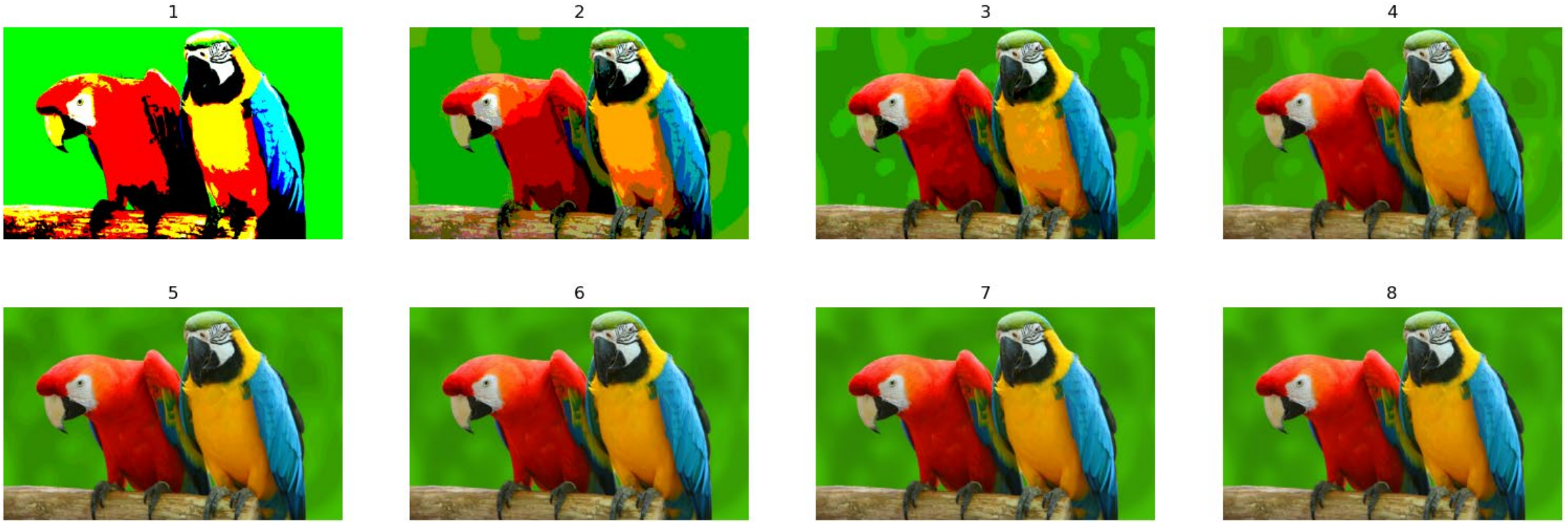
- Spatial resolution

```
n = 3
for i in range(1, n+1):
    plt.subplot(1, n, i)
    s = 1/2**(i+1)
    plt.imshow(cv2.resize(im, None, fx=s, fy=s))
    plt.axis('off'); plt.title(str(s))
plt.show()
```



Resolutions

- Intensity (gray) level resolution: bit per channel



- Need to fix the scale for visualization (8 bpc)

Bit planes

- Convert pixel value into bit
  - 8 bit planes per channel
  - 24 bit planes per colored image

```
def image2bitplane(img):
    bp = []
    for c in range(img.shape[2]):
        for i in range(8):
            bp.append(img[:, :, c] // 2 ** i % 2)
    return np.array(bp)

def bitplane2image(bp):
    img = np.zeros((bp[0].shape[0], bp[0].shape[1], 3))
    for b in range(len(bp)):
        img[:, :, b//8] = img[:, :, b//8] + (bp[b] * 2**(b % 8))
    return np.uint8(img)
```

```
for bpc in range(1, 9):
    bp = image2bitplane(im)
    bp[0:8-bpc, :, :] = 0  # R
    bp[8:8+8-bpc, :, :] = 0  # G
    bp[16:16+8-bpc, :, :] = 0  # B
    im_ = bitplane2image(bp)
    max_vl = sum([2**i for i in range(8-bpc, 8)])
    im_ = np.uint8(im_ / max_vl * 255)
    plt.subplot(2, 4, bpc)
    plt.imshow(im_)
    plt.axis('off')
    plt.title(str(bpc))
plt.show()
```

5 min. challenge II

- Find another image in this image
  - Hint:
    - Use cv2.imread
    - Secret sequence = [19, 18, 17, 16, 11, 10, 9, 8, 3, 2, 1, 0]

challenge_2.png

How to create an video

- Select supported video format: DIVX, XVID, MJPG, X264, WMV1, WMV2, MP4V,

```
import cv2
import numpy as np

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
frame_size = (480, 640, 3)
vdo_writer = cv2.VideoWriter('out.mp4', fourcc, 30, (frame_size[1], frame_size[0]))

for i in range(640):
    frame = np.zeros(frame_size, np.uint8)
    cv2.circle(frame, (i, 200), 20, (255,255,255), -1)
    vdo_writer.write(frame)
vdo_writer.release()
```

How to import a video

- Read frame-by-frame

```
import cv2

cap = cv2.VideoCapture('out.mp4')
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        cv2.imshow('frame', frame)
    if cv2.waitKey(1) == ord('q') or not ret:
        break
cap.release()
cv2.destroyAllWindows()
```

Filename: out.mp4, …

Webcam: 0, 1, 2, …
Use cv2.CAP_DSHOW if it slowly opened

IPcam, NVS: rtsp://u:p@192.168.1.47:554/

Basic Computer vision techniques

- Color Detection
- Frame Difference
- Thresholding
- Morphological Transformations
- Image Segmentation
- Geometric transformations
- Blob analysis

Color Detection

- RGB color space
  - But in OpenCV will be BGR

A

A[:,:,2]

A[:,:,1]

A[:,:,0]

Detect a defined color

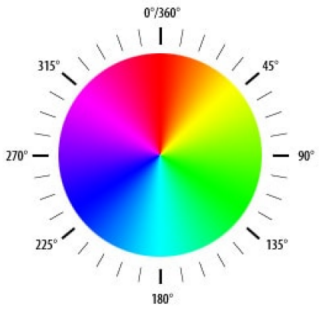
```
import cv2
import numpy as np

A = cv2.imread('parrot.jpg')

tol = 30
color = np.array([64, 116, 138])

mask = cv2.inRange(A, color-tol, color+tol)
Amask = cv2.bitwise_and(A, A, mask=mask)
cv2.imshow('mask', mask)
cv2.imshow('Amask', Amask)
cv2.waitKey()
```

mask

Amask

5 min. challenge III

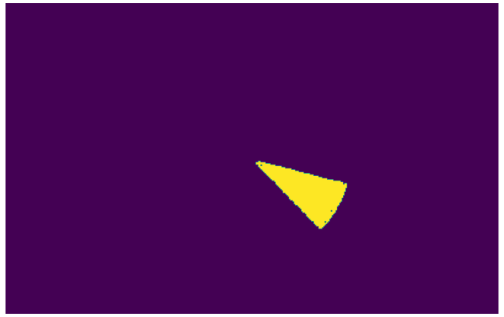
RGB



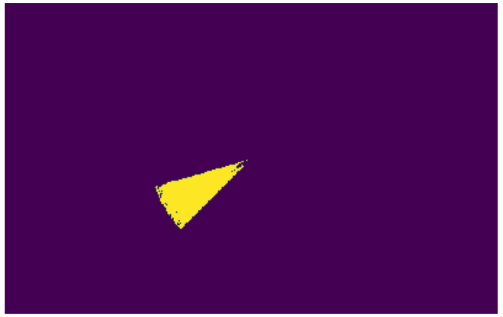
challenge_3.jpg

red

green

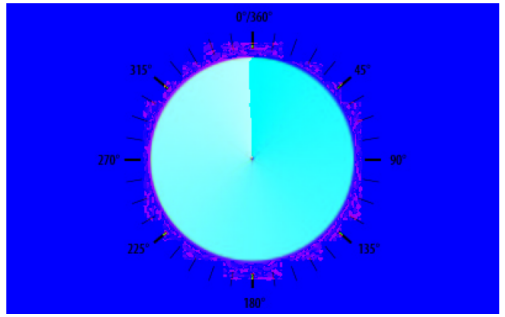


blue

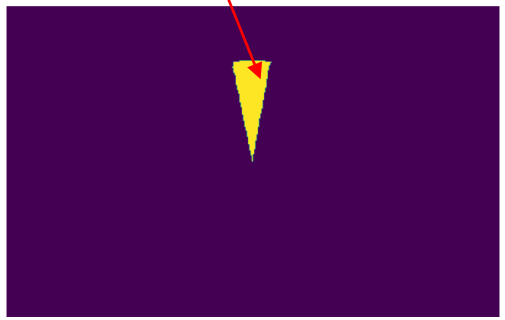


How?

HSV

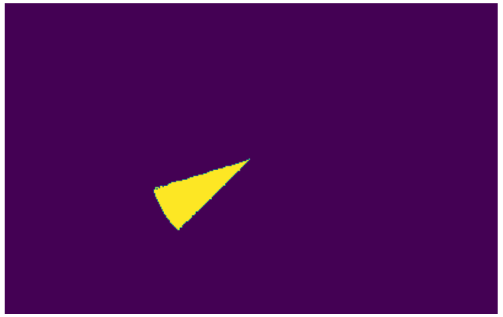


red



green

blue



Background Subtraction

If we have background

Absolute difference

- Beware the data type (uint8)
- Positive or negative value of difference is considered equally

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

im1 = plt.imread('im1.jpg')
im2 = plt.imread('im2.jpg')

diff = cv2.absdiff(im1, im2)
diff = np.uint8(np.abs(im1/255 - im2/255)*255)
```

- HSV for Hue [0, 360]:
  - Opencv [0, 180]
  - Return [0, 1]

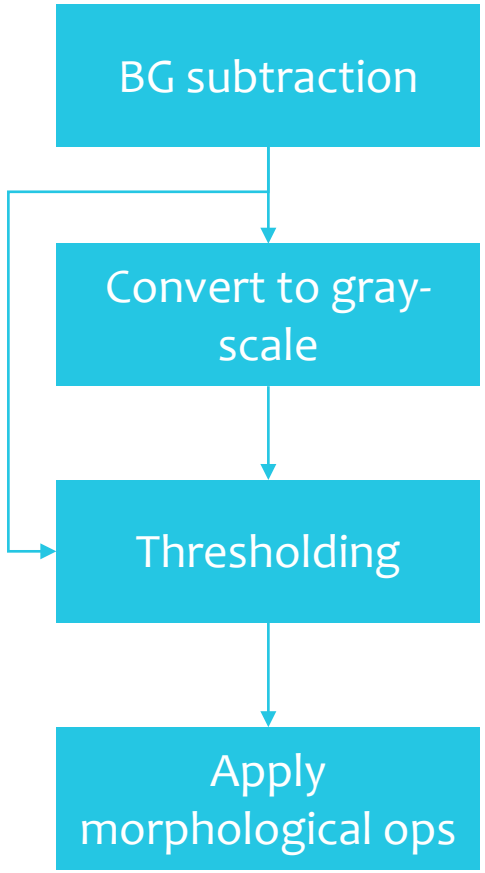
```
def angle_diff(a1, a2):
    return (180 - abs(abs(a1 - a2) - 180)) / 360
```

Absolute difference

diff_RGB

diff_H

Basic image segmentation




```
A = cv2.absdiff(im1,im2)
```

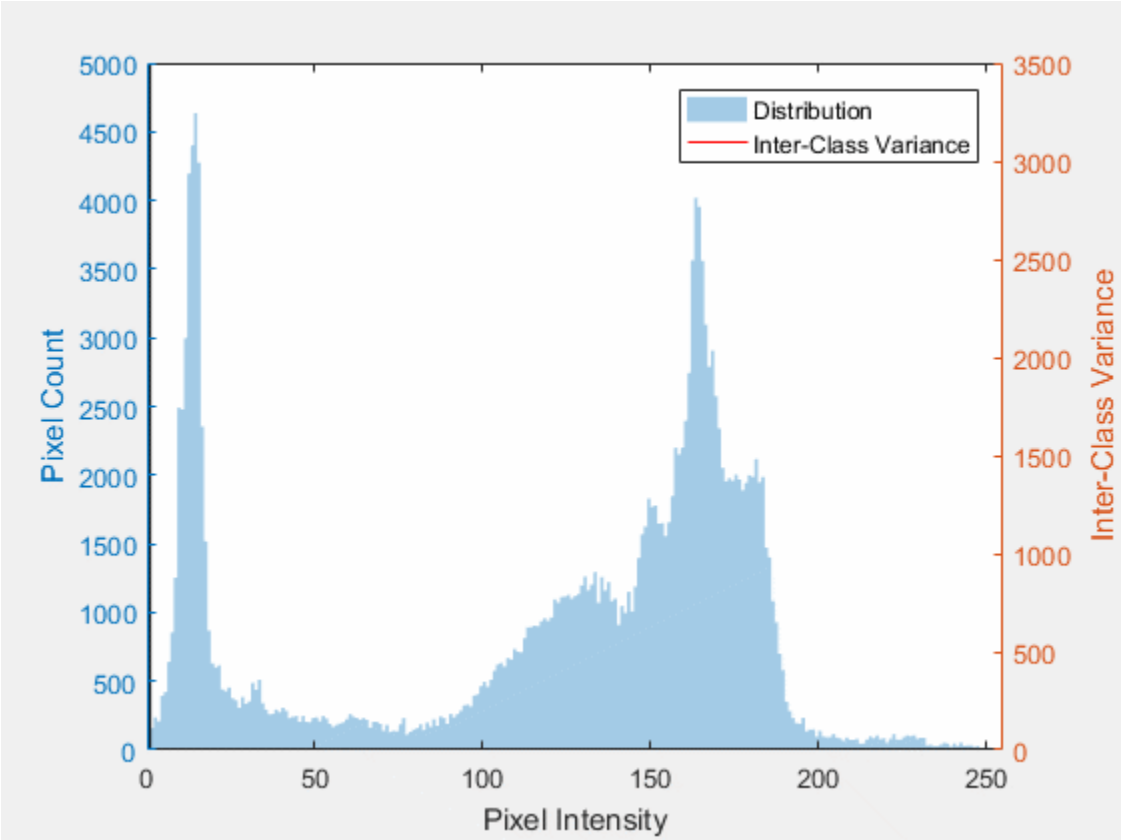
```
G = cv2.cvtColor(A,cv2.COLOR_BGR2GRAY)
```

```
bw = cv2.threshold(G,125,255,cv2.THRESH_BINARY)[1]
bw = cv2.threshold(G,0,255,cv2.THRESH_BINARY|cv2.THRESH_OTSU)[1]
```

```
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(bw,kernel,iterations=1)
dilation = cv2.dilate(bw,kernel,iterations=1)
```

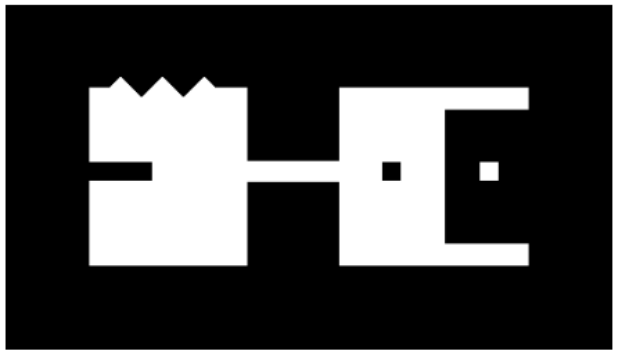
OTSU's method

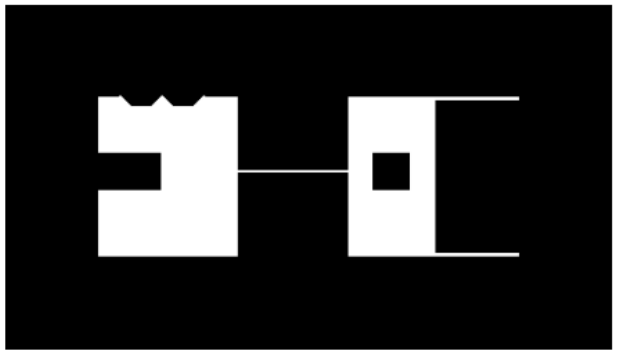
- Assume: The histogram have bimodal distribution
  - Minimize intra-class variance or
  - Maximize inter-class variance



https://en.wikipedia.org/wiki/Otsu%27s_method#/media/File:Otsu's_Method_Visualization.gif

Morphological operations

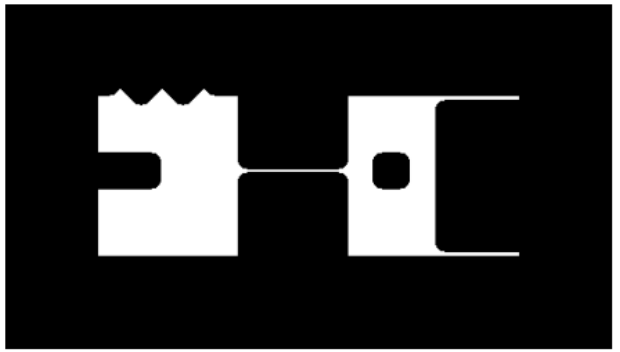




```
kernel = np.ones((17, 17))
```

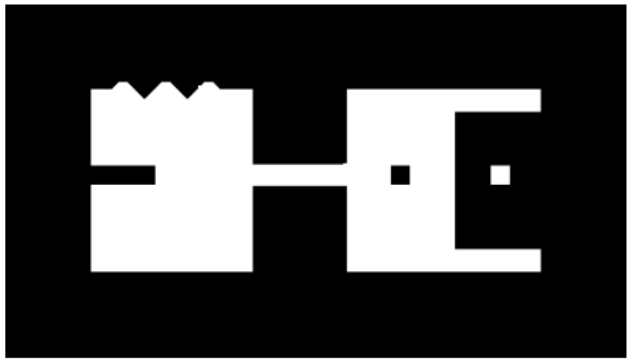
```
cv2.dilate(im, kernel)
```

```
cv2.erode(im, kernel)
```

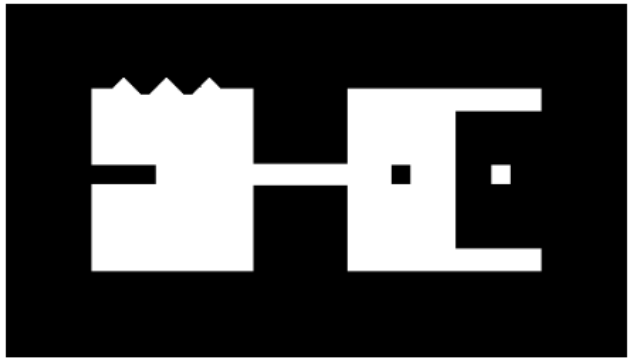


```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17, 17))
```

Morphological operations



Opening = erosion followed by dilation (removing noise)
Closing = dilation followed by erosion (closing small holes)



Remarks

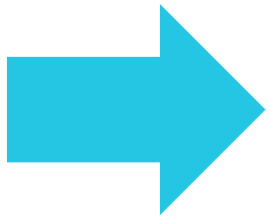
- How to add noise in image
  - Uniform noise: np.random.rand()
  - Gaussian noise: np.random.randn()
  - Salt-and-Pepper: use uniform noise with thresholding
  - Beware the data type
  - EX: add uniform noise (with size)

```
noise = (np.random.rand(*im.shape) > 0.999) + 0.  # float -> bool -> float
noise = cv2.dilate(noise, np.ones((5, 5)))  # bigger size
im1 = im / 255 + noise # float
im1[im1 > 1] = 1  # remove out of range values
im1 = np.uint8(im1*255)  # optional convert to uint8
```

5 min. challenge IV

im1.jpg

im2.jpg



object

Background Subtraction

- If we don't have background
- Non-recursive method
  - |f(t)-f(t-1)| > t
  - |f(t)-mean(f(t-n)… f(t-1))| > t
  - |f(t)-median(f(t-n)… f(t-1))| > t
- Recursive method
  - MOG
  - KNN
  - etc.

BG subtraction: |f(t)-f(t-1)| > t

```
import cv2
import numpy as np

cap = cv2.VideoCapture('cctv.mp4')
buffer = None
while True:
    ret, frame = cap.read()
    if not ret:
        break
    if buffer is not None:
        diff = np.mean(cv2.absdiff(frame, buffer), axis=2)
        fg = np.uint8((diff > 0.5*np.max(diff)) * 255)
        cv2.imshow('diff', np.uint8(diff / np.max(diff) * 255))
        cv2.imshow('fg', fg)
    buffer = frame
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
```

Can use any color space

- Gray, RGB, HSV, etc.

BG subtraction: |f(t)-mean(f(t-n)…f(t-1))| > t

```
import cv2
import numpy as np

cap = cv2.VideoCapture('cctv.mp4')
n_buffer = 10
buffer = None
iframe = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    if buffer is None:
        buffer = np.zeros((n_buffer, *gray.shape))
    buffer[iframe % n_buffer] = gray
    if iframe >= n_buffer:
        M = np.mean(buffer, axis=0)
        diff = cv2.absdiff(gray, np.uint8(M))
        fg = np.uint8((diff > .4 * np.max(diff)) * 255)
        cv2.imshow('fg', fg)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
    iframe += 1
```

Can use any color space

- Gray, RGB, HSV, etc.

BG subtraction: |f(t)-med(f(t-n)…f(t-1))| > t

```python
import cv2
import numpy as np

cap = cv2.VideoCapture('cctv.mp4')
n_buffer = 10
buffer = None
iframe = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    if buffer is None:
        buffer = np.zeros((n_buffer, *gray.shape))
    buffer[iframe % n_buffer] = gray
    if iframe >= n_buffer:
        M = np.median(buffer, axis=0)
        diff = cv2.absdiff(gray, np.uint8(M))
        fg = np.uint8((diff > .4 * np.max(diff)) * 255)
        cv2.imshow('fg', fg)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
    iframe += 1
```

Need sorted data, slow processing

Can use any color space

- Gray, RGB, HSV, etc.

BG subtraction: Recursive method

MOG

- Parametric based algorithm
- Use mixture of Gaussian for modeling each pixel

$$\hat{p}(\vec{x}|\mathcal{X}_T, \text{BG} + \text{FG}) = \sum_{m=1}^{M} \hat{\pi}_m \mathcal{N}(\vec{x}; \hat{\vec{\mu}}_m, \hat{\sigma}_m^2 I)$$

- Approximate the background model by the first B largest clusters

$$\hat{p}(\vec{x}|\mathcal{X}_T, \text{BG}) \sim \sum_{m=1}^{B} \hat{\pi}_m \mathcal{N}(\vec{x}; \hat{\vec{\mu}}_m, \sigma_m^2 I)$$

$$B = \arg\min_b \left(\sum_{m=1}^{b} \hat{\pi}_m > (1 - c_f) \right)$$

KNN

- Non-parametric based algorithm
- Use kernel function for modeling each pixel

$$\hat{p}(\vec{x}|\mathcal{X}_T, \text{BG} + \text{FG}) = \frac{1}{TV} \sum_{m=t-T}^{t} \mathcal{K}\left(\frac{\|\vec{x}^{(m)} - \vec{x}\|}{D} \right) = \frac{k}{TV}$$

- Approximate the background model by the first B largest clusters

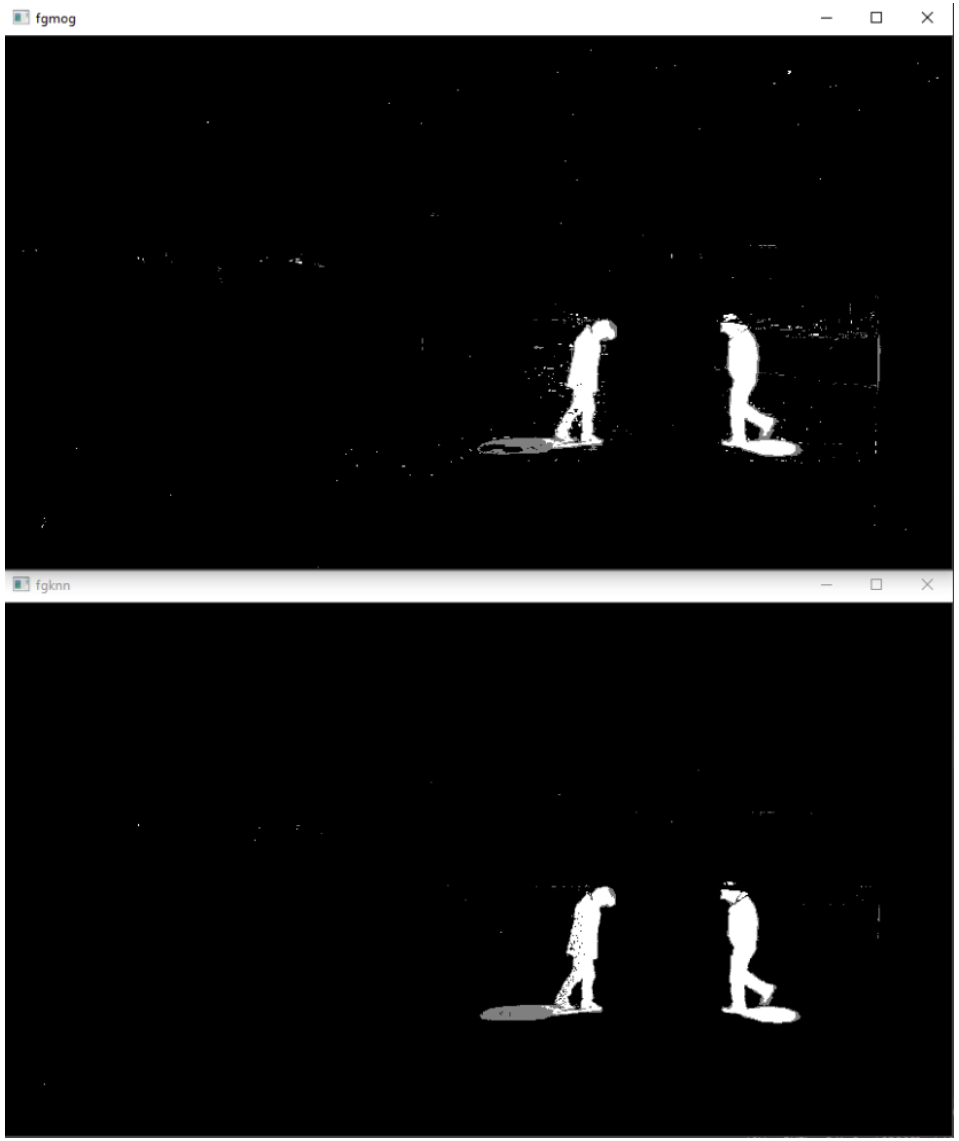
$$\hat{p}(\vec{x}|\mathcal{X}_T, \text{BG}) \approx \frac{1}{TV} \sum_{m=t-T}^{t} b^{(m)} \mathcal{K}\left(\frac{\|\vec{x}^{(m)} - \vec{x}\|}{D} \right)$$

$b^{(m)}$ = 1 if it belongs to background else 0

BG subtraction: Recursive method

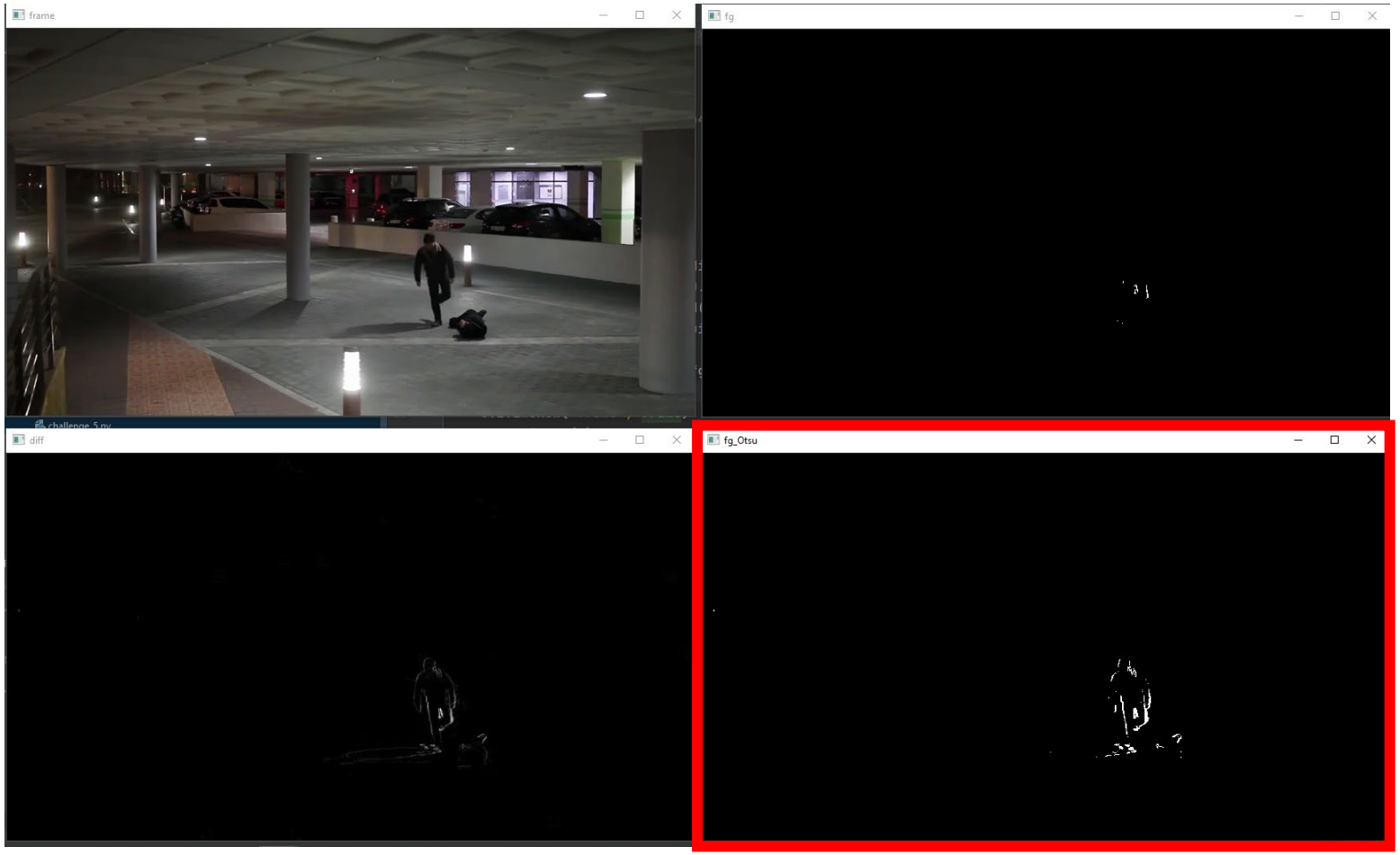
```
import cv2

cap = cv2.VideoCapture('cctv.mp4')
bgsubknn = cv2.createBackgroundSubtractorKNN()
bgsubmog = cv2.createBackgroundSubtractorMOG2()
while True:
    ret, frame = cap.read()
    if not ret:
        break
    fgknn = bgsubknn.apply(frame)
    fgmog = bgsubmog.apply(frame)
    cv2.imshow('frame', frame)
    cv2.imshow('fgknn', fgknn)
    cv2.imshow('fgmog', fgmog)
    cv2.waitKey(1)
```

Normally, KNN perform better than MOG



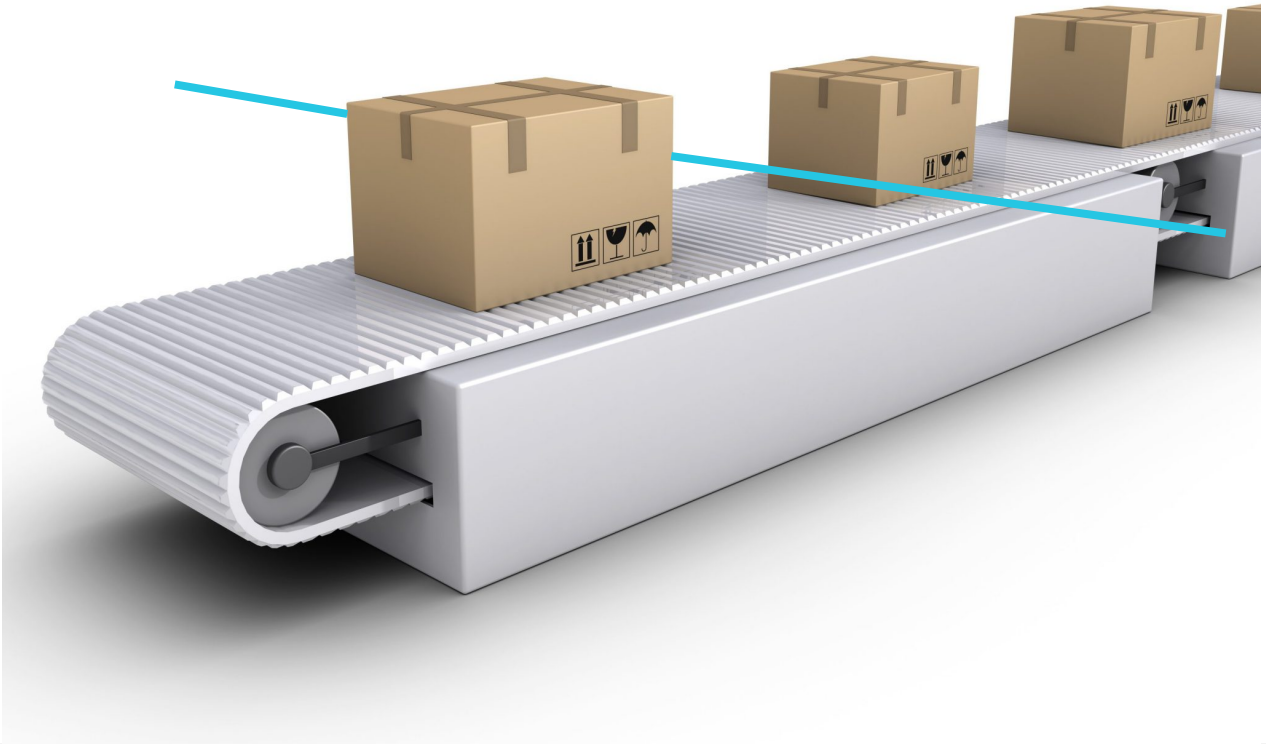
5 min. challenge V

- Use Otsu's method for thresholding in |f(t)-f(t-1)| > t



Line sensor

- Belt conveyor



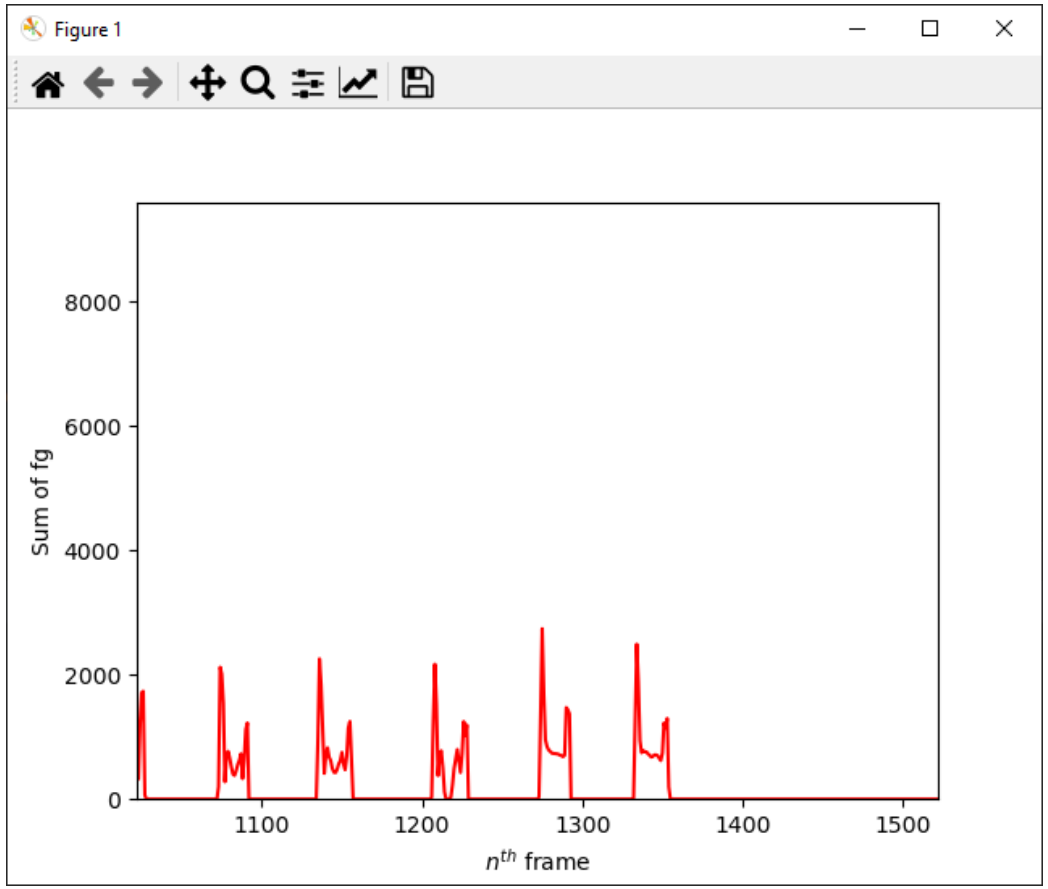
Line sensor

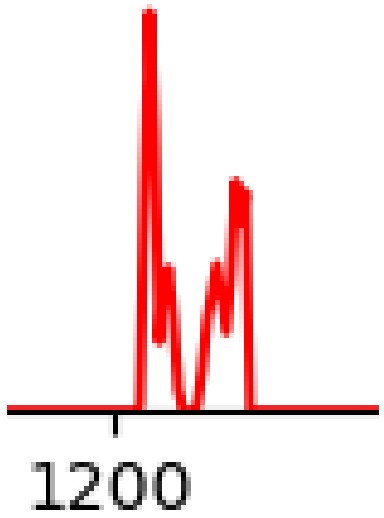
- Use background subtraction in sensor area

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
L1 = np.r_[100:120]
bgsub = cv2.createBackgroundSubtractorKNN()
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame[:, L1, :], cv2.COLOR_BGR2GRAY)
    fg = bgsub.apply(gray)
    if np.sum(fg) > 100:
        frame[:, L1, :2] = 0
    else:
        frame[:, L1, 1:] = 0
    cv2.imshow('fg', fg)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
```

Counting: Line sensor

- Consider the sensor data





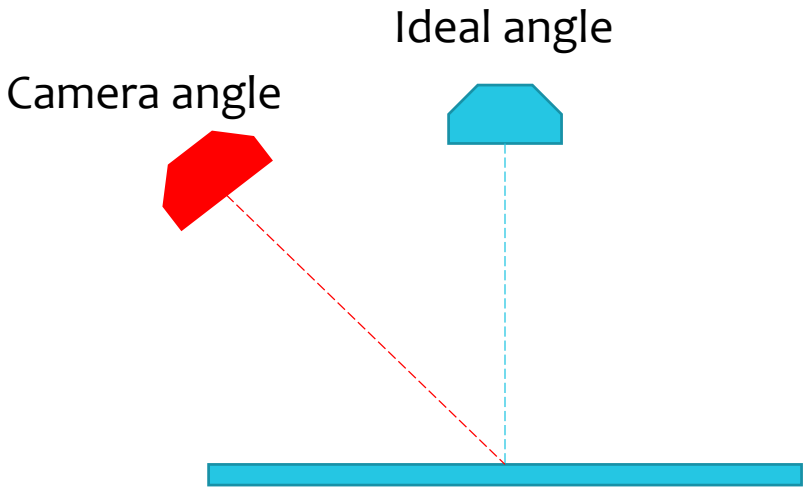
Counting: Line sensor

- Add multiple lines
  - Sensor lines
  - Reset lines

```
import cv2
import numpy as np
cap = cv2.VideoCapture(0)
L1, L2, L3, L4 = np.r_[100:120], np.r_[540:560], np.r_[0:20], np.r_[620:640]
line1, line2, line3, line4 = False, False, False, False
bgsub = cv2.createBackgroundSubtractorKNN(history=1)
count = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    fg = bgsub.apply(gray)
    if np.sum(fg[:, L1]) > 100:
        frame[:, L1, :2] = 0
        line1 = True
    else:
        frame[:, L1, 1:] = 0
    if np.sum(fg[:, L2]) > 100:
        frame[:, L2, :2] = 0
        line2 = True
    else:
        frame[:, L2, 1:] = 0
    if np.sum(fg[:, L3]) > 100 or np.sum(fg[:, L4]) > 100 :
        line1, line2 = False, False
    if line2 and line1:
        line1, line2 = False, False
        count += 1
    cv2.putText(frame, str(count), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow('fg', fg)
    cv2.imshow('frame', frame)
    cv2.waitKey(1)
```

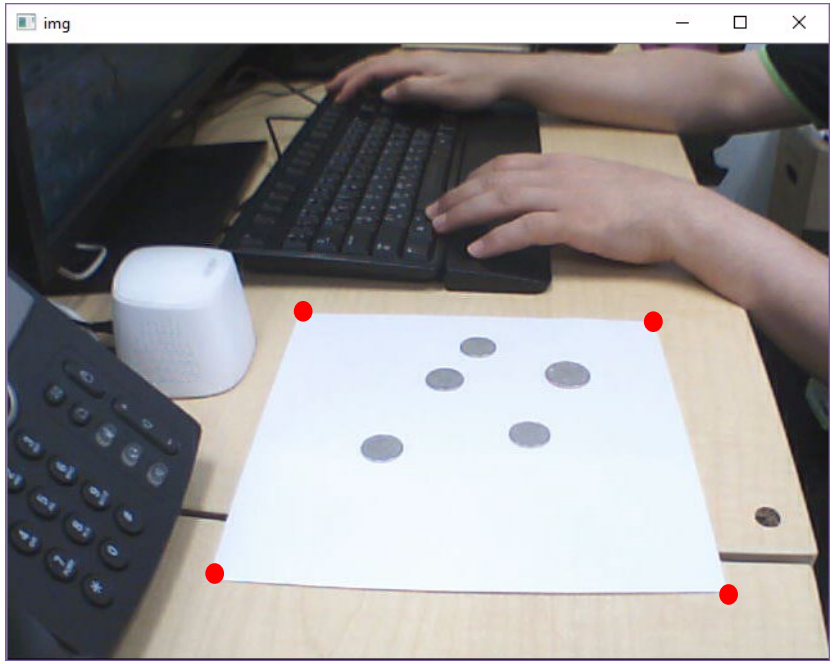
Perspective transformation

- The best viewing angle for this application is top view but sometimes we cannot set camera at this angle



Perspective Transform

- Object is smaller as its distance from camera increases
- Find 4 points to estimate the transformation (same sequence)




```
T = cv2.getPerspectiveTransform(np.float32(pts1),np.float32(pts2))
img = cv2.warpPerspective(img, T, template_size)
```

Point selecting with mouse click

- Define a callback that corresponding with display window

```
def onClick(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        ...


cv2.namedWindow('img')
cv2.setMouseCallback('img', onClick)
```

Blob analysis

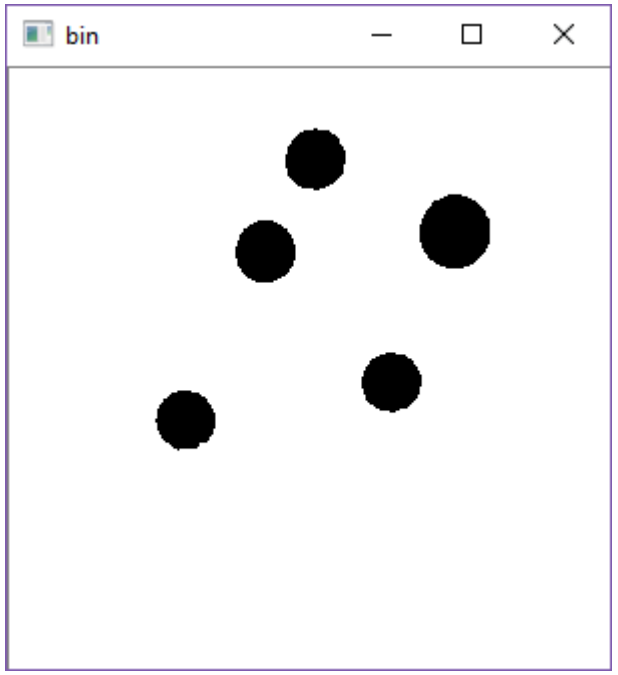
- cv2.findContours() for binary image
  - Connected points are grouped together as object
  - Find object with suitable properties, e.g. find the smallest circle that can cover this object

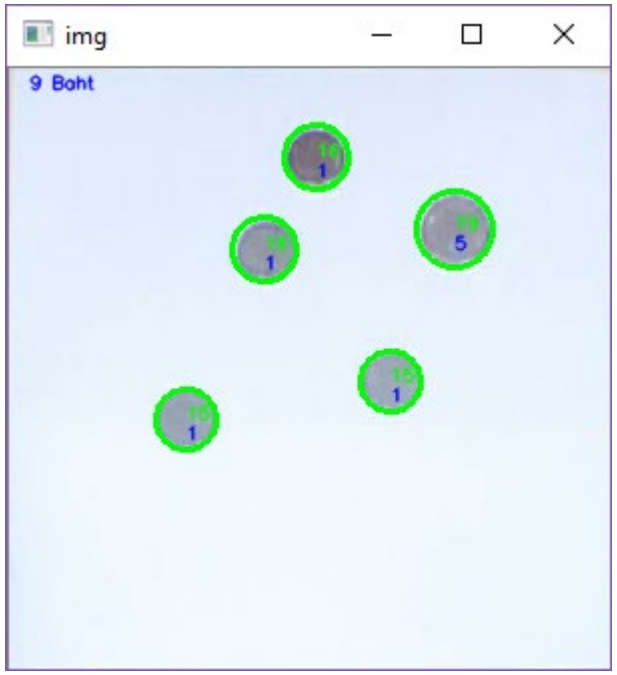
```
contours, hierarchy = cv2.findContours(bin, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
for cnt in contours:
    (x, y), radius = cv2.minEnclosingCircle(cnt)
    center = (int(x), int(y))
    radius = int(radius)
    img = cv2.circle(img, center, radius, (0, 255, 0), 2)
```

- The number of contours is huge because of noise
  - Add preprocessing with **erode** for denoising and **dilate** to maintain the object size

```
kernel = np.ones((11,11))
bw = cv2.erode(bw, kernel)
bw = cv2.dilate(bw, kernel)
```

Coin Counting






```
if 14 <= radius <= 17:
    img = cv2.putText(img, '1', (int(x), int(y) + 10), font, .3, (255, 0, 0), 1, cv2.LINE_AA)
    count += 1
if 18 <= radius <= 20:
    img = cv2.putText(img, '5', (int(x), int(y) + 10), font, .3, (255, 0, 0), 1, cv2.LINE_AA)
    count += 5
```

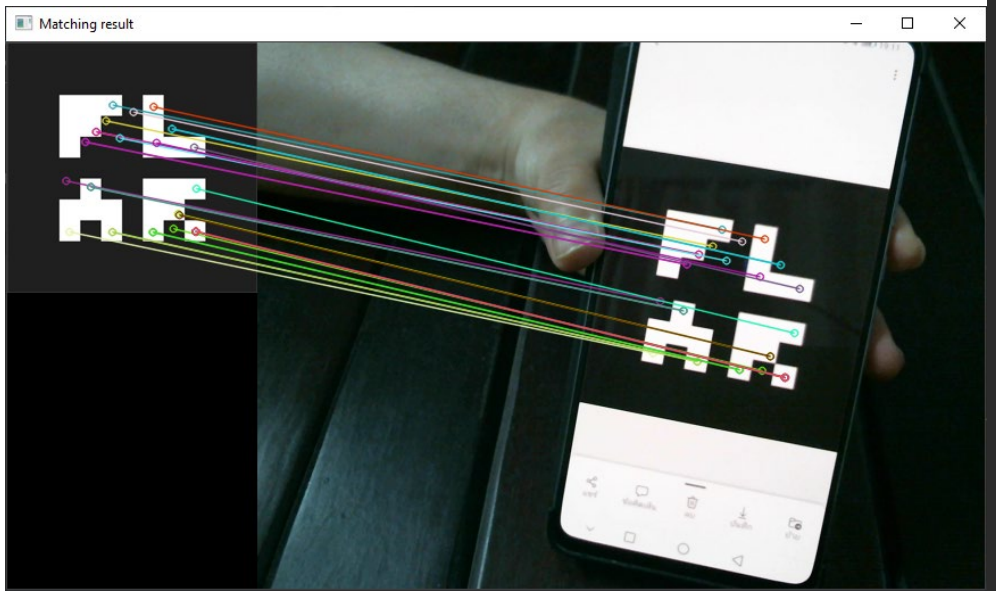
Feature Detection

- Scale-invariant feature transform: SIFT
  - Output
    - Keypoint: location, angle, size, etc.
    - Descriptor: 128-D vector of each keypoint
      - Use for similarity measurement


```
import cv2

img1 = cv2.imread("ARmarker.png", cv2.IMREAD_GRAYSCALE)
ft = cv2.SIFT_create()
kp, des = ft.detectAndCompute(img1, None)
img = cv2.drawKeypoints(img1, kp, img1, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
cv2.imshow('img', img)
cv2.waitKey()
```

Feature Matching

- Brute-Force
- FLANN (Fast Library for Approximate Nearest Neighbors)
  - Randomized kd-tree k-nn




```python
import cv2
import numpy as np

img1 = cv2.imread("ARmarker.png", cv2.IMREAD_GRAYSCALE)
ft = cv2.SIFT_create()
kp1, des1 = ft.detectAndCompute(img1, None)
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)

# FLANN parameters
index_params = dict(algorithm=1, trees=5)
search_params = dict(checks=50)
flann = cv2.FlannBasedMatcher(index_params, search_params)

while True:
    ret, img2 = cap.read()
    kp2, des2 = ft.detectAndCompute(img2, None)
    matches = flann.knnMatch(des1.astype(np.float32), des2.astype(np.float32), k=2)
    good = []
    for m, n in matches:
        if m.distance < 0.6*n.distance:
            good.append(m)
    matching_result = cv2.drawMatches(img1, kp1, img2, kp2, good[:50], None, flags=2)
    cv2.imshow("Matching result", matching_result)
    cv2.waitKey(1)
```

Ex: Augmented Reality

- Use SIFT for detect location and position of AR core
- Use FLANN for feature matching
- Use Perspective transformation for mapping an image onto the AR core

Object tracking

- Initial object location
- Find object in search space (lower than original space)

- Standard OpenCV 4.5.1:
  - MIL
  - GOTURN (CNN-based)
    - Pretrained: https://github.com/spmallick/goturn-files

Big challenge

- Create game recording for checker (Thai rule)
  - Create demonstration video and submit your video link