# Practical 1

## B. Perform Image Stitching

**Code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

img\_= cv2.imread('left\_img.jpg')

img1= cv2.cvtColor(img\_, cv2.COLOR\_BGR2GRAY)

img= cv2.imread('right\_img.jpg')

img2= cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

sift=cv2.SIFT\_create()

kp1, des1= sift.detectAndCompute(img1, None)

kp2, des2= sift.detectAndCompute(img2, None)

bf= cv2.BFMatcher(cv2.NORM\_L2, crossCheck=False)

matches=bf.knnMatch(des1,des2,k=2)

good=[]

for m, n in matches:

if m.distance < 0.75\* n.distance:

good.append(m)

if len(good) > 4:

src\_pts = np.float32([kp1[m.queryIdx].pt for m in good]).reshape(-1,1,2)

dst\_pts = np.float32([kp2[m.trainIdx].pt for m in good]).reshape(-1,1,2)

H, status = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0)

dst=cv2.warpPerspective(img\_, H, (img.shape[1] + img\_.shape[1] + img.shape[0]))

dst\_rgb = cv2.cvtColor(dst, cv2.COLOR\_BGR2GRAY)

plt.subplot(122), plt.imshow(dst\_rgb), plt.title('Warped Image')

plt.show()

dst[0:img.shape[0], 0:img.shape[1]]= img

combined\_rgb = cv2.cvtColor(dst, cvr.COLOR\_BGR2GRAY)

cv2.imwrite('Output.jpg', dst)

plt.imshow(combined\_rgb)

plt.title('Stitched Image')

plt.show()

else:

raise AssertionError("Not enough matches are found - {}/{}".format(len(good), 4))

# Practical 2

## Face Detection

**Image Face Detection:**

!pip install opencv-python

import cv2

face\_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascades\_frontalface\_default.xml')

image\_path="img.jfif"

image=cv2.imread(image\_path)

gray\_image=cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

faces=face\_cascade.detectMultiScale(gray\_image, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

for (x,y,w,h) in faces:

cv2.rectangle(image,(x,y),(x+w,y+h), (255,0,0),2)

output\_path='facesdetecetd.jpg'

cv2.imwrite(output\_path, image)

print('Faces detected: {len(faces)}. Output saved to {output\_path}')

**Live Face Detection:**

import cv2

import matplotlib.pyplot as plt

face\_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade\_frontalface\_default.xml')

cap=cv2.VideoCapture(0)

ret, frame=cap.read()

if ret:

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

faces=face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

#Draw rectangles around detected faces

for (x, y, w, h) in faces:

cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 2)

plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

plt.axis('off') #Turn off axis Labels

plt.show()

#Release capture

cap.release()

print("Number faces detected:", len(faces))

## Object Detection

**Code:**

import cv2

import argparse

import numpy as np

args = argparse.Namespace(image="dog.jpg", config="yolov3.cfg", weights="yolov3.weights", classes="yolov3.txt")

def get\_output\_layers(net):

layer\_names = net.getLayerNames()

try:

output\_layers = [layer\_names[i-1] for i in net.getUnconnectedOutLayers()]

except:

output\_layers = [layer\_names[i[0]-1] for i in net.getUnconnectedOutLayers()]

return output\_layers

def draw\_prediction(img, class\_id, confidence, x, y, x\_plus\_w, y\_plus\_h):

label = str(classes[class\_id])

color = COLORS[class\_id]

cv2.rectangle(img, (x,y), (x\_plus\_w, y\_plus\_h), color, 2)

cv2.putText(img, label, (x-10, y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

image = cv2.imread(args.image)

Width = image.shape[1]

Height=image.shape[0]

scale = 0.00392

classes = None

with open(args.classes, 'r') as f:

classes = [line.strip() for line in f.readlines()]

COLORS = np.random.uniform(0, 255, size=(len(classes), 3))

net = cv2.dnn.readNet(args.weights, args.config)

blob = cv2.dnn.blobFromImage(image, scale, (416,416), (0,0,0), True, crop=False)

net.setInput(blob)

outs = net.forward(get\_output\_layers(net))

class\_ids = []

confidence = []

boxes = []

conf\_threshold = 0.5

nms\_threshold = 0.4

for out in outs:

for detection in out:

scores = detection[5:]

class\_id = np.argmax(scores)

confidences = scores[class\_id]

if confidence > 0.5:

center\_x = int(detection[0] \* Width)

center\_y = int(detection[1] \* Height)

w = int(detection[2] \* Width)

h = int(detection[3] \* Height)

x = center\_x - w/2

y = center\_y - h/2

class\_ids.append(class\_id)

confidences.append(float(confidence))

boxes.append([x, y, w, h])

indices = cv2.dnn.NMSBoxes(boxes, confidences, conf\_threshold, nms\_threshold)

for i in indices:

try:

box = boxes[i]

except:

i = i[0]

box = boxes[i]

x= box[0]

y= box[1]

w= box[2]

h= box[3]

draw\_prediction(image, class\_ids[i], confidences[i], round(x), round(y), round(x+w), round(y+h))

import matplotlib.pyplot as plt

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.title('Object Detection')

plt.axis('off')

plt.show()

cv2.imwrite('object-detection.jpg', image)

## Pedestrian detection

**Method 1**

!pip install opencv-python

!pip install matplotlib

import cv2

import matplotlib.pyplot as plt

pedestrian\_cascade=cv2.CascadeClassifier(cv2.data.haarcascades+'haarcascade\_fullbody.xml')

video\_path='VIDEO.mp4'

cap=cv2.VideoCapture(video\_path)

if not cap.isOpened():

print("Error: Unable to open the video")

exit()

while True:

ret, frame = cap.read()

if not ret:

break

gray\_frame=cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

pedestrians=pedestrian\_cascade.detectMultiScale(gray\_frame, scaleFactor=1.1, minNeighbors=5, minSize=(30,30))

for (x,y,w,h) in pedestrians:

cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)

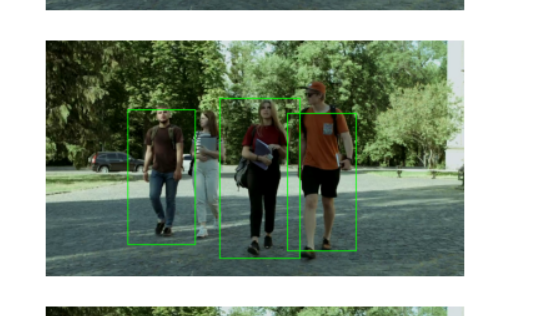
plt.imshow(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()

cap.release()





**Another Method:**

import cv2

import matplotlib.pyplot as plt

import matplotlib.animation as animation

#Initialize the Haar Cascade pedestrian detection model

pedestrian\_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_fullbody.xml')

# Load the video

video\_path = 'D:/VIDEO.mp4'

cap= cv2.VideoCapture(video\_path)

#check if the video is opened successfully

if not cap.isOpened():

print("Error: Unable to open the video.")

exit()

#Function to detect pedestrians and draw bounding boxes

def detect\_pedestrians(frame):

#Convert the frame to grayscale

gray\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

#Detect pedestrians in the grayscale frame

pedestrians = pedestrian\_cascade.detectMultiScale(gray\_frame, scaleFactor=1.05,minNeighbors=5,minSize=(30,30))

#Draw rectangles around the detected pedestrains

for(x,y,w,h) in pedestrians:

cv2.rectangle( frame,(x,y),(x+w,y+h),(0,255,0),2)

return frame

fig,ax=plt.subplots()

#Function to update the animation

def update(frame):

#Read a frME FROM THE VIDEO

ret,frame = cap.read()

if not ret:

ani.event\_source.stop()

return

#Detect pedestrians and draw boundng boxes

frame\_with\_pedestrains=detect\_pedestrians(frame)

ax.clear()

ax.imshow(cv2.cvtColor(frame\_with\_pedestrains, cv2.COLOR\_BGR2RGB))

ax.axis('off')

#create the animation

ani=animation.FuncAnimation(fig,update,interval=50)

#Display the animation

plt.show()

#Release the video capture object

cap.release

