# SMART TRAFFIC SIGNAL MANAGEMENT USING PYTHON CODING



TEAM MEMBERS…

D.Milwin Antany(821021104030)

A.Vishal(821021104701)

R.Logesh Raj(821021104029)

R.Nasren Cruze(821021104033)

K.Vasanth(821021104306)

There has been a population increase which has consequently led to traffic congestion in the city of Karachi. Making a smart traffic management system that makes use of video and picture data of the traffic on the roads of Karachi, Pakistan. This works by performing machine learning using an algorithm over the recent frame obtained from the video to estimate the number of vehicles present in a scene. Cameras will be installed on the opposite of the lane, beside the traffic light and will take its real-time video. At the back-end, Raspberry Pi would be connected to handle video processing. Raspberry pi would receive video as input from the camera of each road. Image framing would capture frames from the video at several fixed intervals. By taking our city, Karachi, into consideration we are creating our data set based on images captured from within the city. The proposed project aims to make decisions for the traffic signal timings based on vehicle densities. The project will be deployed on a four-way traffic signal. It will make use of image processing to separate image frames while machine learning algorithms will perform the task of signal controlling and vehicle detection. The reason for using Image Processing and machine learning is because it keeps production costs are low while achieving high speed and accuracy.

# CODING :

import os

def remove\_img(folder):

folder\_path = (folder)

test = os.listdir(folder\_path)

for images in test:

if images.endswith(".jpg"):

os.remove(os.path.join(folder\_path, images))

# USING PYTHON CODING :

import numpy as np

import time

import cv2

import os

import re

import vctrl

import removing\_func

import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)

GPIO.setwarnings(False)

#setting pins for lights

r1 = 37

y1 = 35

g1 = 33

r2=31

y2=29

g2=23

r3=21

y3=19

g3=15

r4=13

y4=11

g4=7

#setting pin functions

GPIO.setup(r1, GPIO.OUT)

GPIO.setup(y1, GPIO.OUT)

GPIO.setup(g1, GPIO.OUT)

GPIO.setup(r2, GPIO.OUT)

GPIO.setup(y2, GPIO.OUT)

GPIO.setup(g2, GPIO.OUT)

GPIO.setup(r3, GPIO.OUT)

GPIO.setup(y3, GPIO.OUT)

GPIO.setup(g3, GPIO.OUT)

GPIO.setup(r4, GPIO.OUT)

GPIO.setup(y4, GPIO.OUT)

GPIO.setup(g4, GPIO.OUT)

#initial default state

GPIO.output(r1, True)

GPIO.output(y1, False)

GPIO.output(g1, False)

GPIO.output(r2, True)

GPIO.output(y2, False)

GPIO.output(g2, False)

GPIO.output(r3, True)

GPIO.output(y3, False)

GPIO.output(g3, False)

GPIO.output(r4, False)

GPIO.output(y4, False)

GPIO.output(g4, True)

#default initial green light delay

green=1

#default yellow light delay

yellow=2

#lane to start with

p=1

#for system to work infinite times

while(1):

#for not exceeding 4 lanes

while(p<5):

l=0

i = 0

k = 0

#defining all required parameters for each lane

if p==1:

videoname=2

x=300

w=600

y=0

h=400

folder="images1/"

elif p==2:

videoname=2

x=0

w=300

y=0

h=400

folder="images2/"

elif p==3:

videoname=0

x=300

w=600

y=0

h=400

folder="images3/"

elif p==4:

videoname=0

x=0

w=300

y=0

h=400

folder="images4/"

#emptying the folder before writing in any image

removing\_func.remove\_img(folder)

#capturing video

cap = cv2.VideoCapture(videoname)

#setting resolution

cap.set(3, 1280)

cap.set(4, 720)

#every second frame of the video is to be captured

frame\_no=2

#duration until camera will work, green is the delay of previos lane

capture\_duration = green

start\_time = time.time()

#for reading

col\_images = []

col\_frames = os.listdir(folder)

# sort file names

col\_frames.sort(key=lambda f: int(re.sub('\D', '', f)))

#incase of zero vehicles

if green==0:

capture\_duration=1

elif green-2<0:

capture\_duration=1

elif green-2==0:

capture\_duration=1

else:

capture\_duration=green-2

#capturing video

while ( int(time.time() - start\_time) < capture\_duration ):

ret, frame = cap.read()

if ret == False:

break

else:

#checking if it is the second frame

l=(i % frame\_no == 0)

print("l: "+str(l))

if l==True:

k += 1

frame = cv2.resize(frame, (600, 400))

cv2.imwrite(os.path.join(folder, str(k) + '.jpg'), frame)

print(frame.shape)

# append the frames to the list

frame = cv2.resize(frame, (600, 400))

print(frame.shape)

frame=frame[y:h,x:w]

print(frame.shape)

col\_images.append(frame)

print(k)

i += 1

print(k)

#print("--- %s seconds ---" % (time.time() - start\_time))

# load the COCO class labels our YOLO model was trained on

labelsPath = "classes.names"

LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label

np.random.seed(42)

COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),dtype="uint8")

# paths to the YOLO weights and model configuration

weightsPath = "yolov4final.weights"

configPath = "yolov4testing.cfg"

# load our YOLO object detector trained on COCO dataset (80 classes)

net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# load our input image and grab its spatial dimensions

image = col\_images[k-1]

#image=image[x:w,y:h]

(H, W) = image.shape[:2]

# determine only the output layer names that we need from YOLO

ln = net.getLayerNames()

ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),swapRB=True, crop=False)

net.setInput(blob)

start = time.time()

layerOutputs = net.forward(ln)

end = time.time()

# initialize our lists of detected bounding boxes, confidences, and

# class IDs, respectively

boxes = []

confidences = []

classIDs = []

# loop over each of the layer outputs

for output in layerOutputs:

for detection in output:

scores = detection[5:]

classID = np.argmax(scores)

confidence = scores[classID]

# filter out weak predictions by ensuring the detected

# probability is greater than the minimum probability

if confidence > 0.5:

# scale the bounding box coordinates back relative to the

# size of the image, keeping in mind that YOLO actually

# returns the center (x, y)-coordinates of the bounding

# box followed by the boxes' width and height

box = detection[0:4] \* np.array([W, H, W, H])

(centerX, centerY, width, height) = box.astype("int")

# use the center (x, y)-coordinates to derive the top and

# and left corner of the bounding box

x = int(centerX - (width / 2))

y = int(centerY - (height / 2))

# update our list of bounding box coordinates, confidences,

# and class IDs

boxes.append([x, y, int(width), int(height)])

confidences.append(float(confidence))

classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping bounding

# boxes

idxs = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.3)

count=0

t=0

print('Types of objects detected in image:')

# ensure at least one detection exists

if len(idxs) > 0:

# loop over the indexes we are keeping

for i in idxs.flatten():

t+=1

# extract the bounding box coordinates

(x, y) = (boxes[i][0], boxes[i][1])

(w, h) = (boxes[i][2], boxes[i][3])

# draw a bounding box rectangle and label on the image

color = [int(c) for c in COLORS[classIDs[i]]]

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

text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])

print(str(t)+') '+LABELS[classIDs[i]])

cv2.putText(image, text, (x, y - 5), cv2.FONT\_HERSHEY\_SIMPLEX,0.5, color, 2)

if LABELS[classIDs[i]]=='car':

count=count+1

#print("Number of vehicles= "+ str(count))

# show the output image

#print("--- %s seconds ---" % (time.time() - start\_time))

cv2.imshow("Image", image)

cv2.waitKey(0)

green=vctrl.delays(count)

#lights

if green==0:

if p==1:

##yellow

GPIO.output(r1, False)

GPIO.output(y1, True)

GPIO.output(g1, False)

GPIO.output(r4, False)

GPIO.output(y4, True)

GPIO.output(g4, False)

time.sleep(yellow)

GPIO.output(r4, True)

GPIO.output(y4, False)

GPIO.output(g4, False)

elif p==2:

##yellow

GPIO.output(r2, False)

GPIO.output(y2, True)

GPIO.output(g2, False)

GPIO.output(r1, False)

GPIO.output(y1, True)

GPIO.output(g1, False)

time.sleep(yellow)

GPIO.output(r1, True)

GPIO.output(y1, False)

GPIO.output(g1, False)

elif p==3:

##yellow

GPIO.output(r2, False)

GPIO.output(y2, True)

GPIO.output(g2, False)

GPIO.output(r3, False)

GPIO.output(y3, True)

GPIO.output(g3, False)

time.sleep(yellow)

GPIO.output(r2, True)

GPIO.output(y2, False)

GPIO.output(g2, False)

elif p==4:

##yellow

GPIO.output(r4, False)

GPIO.output(y4, True)

GPIO.output(g4, False)

GPIO.output(r3, False)

GPIO.output(y3, True)

GPIO.output(g3, False)

time.sleep(yellow)

GPIO.output(r3, True)

GPIO.output(y3, False)

GPIO.output(g3, False)

#setting lights according to the lane

else:

if p==1:

##yellow

GPIO.output(r1, False)

GPIO.output(y1, True)

GPIO.output(g1, False)

GPIO.output(r4, False)

GPIO.output(y4, True)

GPIO.output(g4, False)

time.sleep(yellow)

##green

GPIO.output(r1, False)

GPIO.output(y1, False)

GPIO.output(g1, True)

GPIO.output(r2, True)

GPIO.output(y2, False)

GPIO.output(g2, False)

GPIO.output(r3, True)

GPIO.output(y3, False)

GPIO.output(g3, False)

GPIO.output(r4, True)

GPIO.output(y4, False)

GPIO.output(g4, False)

print(p)

time.sleep(green)

elif p==2:

##yellow

GPIO.output(r2, False)

GPIO.output(y2, True)

GPIO.output(g2, False)

GPIO.output(r1, False)

GPIO.output(y1, True)

GPIO.output(g1, False)

time.sleep(yellow)

##green

GPIO.output(r2, False)

GPIO.output(y2, False)

GPIO.output(g2, True)

GPIO.output(r1, True)

GPIO.output(y1, False)

GPIO.output(g1, False)

GPIO.output(r3, True)

GPIO.output(y3, False)

GPIO.output(g3, False)

GPIO.output(r4, True)

GPIO.output(y4, False)

GPIO.output(g4, False)

print(p)

time.sleep(green)

elif p==3:

##yellow

GPIO.output(r2, False)

GPIO.output(y2, True)

GPIO.output(g2, False)

GPIO.output(r3, False)

GPIO.output(y3, True)

GPIO.output(g3, False)

time.sleep(yellow)

##green

GPIO.output(r3, False)

GPIO.output(y3, False)

GPIO.output(g3, True)

GPIO.output(r1, True)

GPIO.output(y1, False)

GPIO.output(g1, False)

GPIO.output(r2, True)

GPIO.output(y2, False)

GPIO.output(g2, False)

GPIO.output(r4, True)

GPIO.output(y4, False)

GPIO.output(g4, False)

print(p)

time.sleep(green)

elif p==4:

##yellow

GPIO.output(r4, False)

GPIO.output(y4, True)

GPIO.output(g4, False)

GPIO.output(r3, False)

GPIO.output(y3, True)

GPIO.output(g3, False)

time.sleep(yellow)

##green

GPIO.output(r4, False)

GPIO.output(y4, False)

GPIO.output(g4, True)

GPIO.output(r1, True)

GPIO.output(y1, False)

GPIO.output(g1, False)

GPIO.output(r2, True)

GPIO.output(y2, False)

GPIO.output(g2, False)

GPIO.output(r3, True)

GPIO.output(y3, False)

GPIO.output(g3, False)

time.sleep(green)

print("P before changing"+str(p))

p+=1

print("P after changing"+str(p))

removing\_func.remove\_img(folder)

cap.release()

if p==5:

p=1

# TRAFFIC SYSTEM CODING :

def delays(count):

i=count

low=10

medium=20

high=30

#super\_high=40

delay=0

print("Number of vehicles= "+ str(i))

if i < low:

if i<1:

print('green light delay = 0 sec')

elif i<4:

delay=i\*1.5

print('green light delay = '+str(delay)+' sec')

elif i<7:

delay=5\*1.5

print('green light delay = '+str(delay)+' sec')

else:

delay=7\*1.5

print('green light delay = '+str(delay)+' sec')

elif i < medium:

if i<medium/2:

delay=i\*1.5

print('green light delay = '+str(delay)+' sec')

else:

delay=i\*1.5

print('green light delay = '+str(delay)+' sec')

elif i < high:

if i<high/2:

delay=i\*1.5

print('green light delay = '+str(delay)+' sec')

else:

delay=60

print('green light delay = '+str(delay)+' sec')

return(delay)

