

Folder: Asphalt

File: color.py

Documented code for color.py:

```
import cv2
```

```
import numpy as np
```

```
cam = cv2.VideoCapture(0)
```

```
cv2.namedWindow('Color Detection')
```

```
def window(x):
```

```
    pass
```

```
cv2.createTrackbar('Hue', 'Color Detection' , 0, 179, window)
```

```
cv2.createTrackbar('Saturation','Color Detection', 0 ,255,window)
```

```
cv2.createTrackbar('Value','Color Detection', 0,255,window)
```

```
while True:
```

```
    ret ,img = cam.read()
```

```
    img = np.flip(img ,axis =1)
```

```
    img = cv2.resize(img ,(480 ,360))
```

```
    hsv = cv2.cvtColor(img ,cv2.COLOR_BGR2HSV)
```

```
    blurred = cv2.GaussianBlur(hsv, (11,11),0)
```

```
    h = cv2.getTrackbarPos('Hue', 'Color Detection')
```

```
    s = cv2.getTrackbarPos('Saturation', 'Color Detection')
```

```
    v = cv2.getTrackbarPos('Value', 'Color Detection')
```

```
lower_color = np.array([h ,s, v])  
upper_color = np.array([100,255,255])  
mask = cv2.inRange(hsv , lower_color,upper_color)  
cv2.imshow ('Color Detection', cv2.bitwise_and(img ,img ,mask=mask))
```

```
key = cv2.waitKey(1) & 0xFF  
if key == ord("q"):  
    break
```

```
cam.release()  
cv2.destroyAllWindows()  
#31 33 153
```

File: directkeys.py

Documented code for directkeys.py:

```
import ctypes  
import time
```

```
SendInput = ctypes.windll.user32.SendInput
```

A = 0x1E

D = 0x20

Space = 0x39

C struct redefinitions

PUL = ctypes.POINTER(ctypes.c_ulong)

class KeyBdInput(ctypes.Structure):

```
_fields_ = [("wVk", ctypes.c_ushort),  
            ("wScan", ctypes.c_ushort),  
            ("dwFlags", ctypes.c_ulong),  
            ("time", ctypes.c_ulong),  
            ("dwExtraInfo", PUL)]
```

class HardwareInput(ctypes.Structure):

```
_fields_ = [("uMsg", ctypes.c_ulong),  
            ("wParamL", ctypes.c_short),  
            ("wParamH", ctypes.c_ushort)]
```

class MouseInput(ctypes.Structure):

```
_fields_ = [("dx", ctypes.c_long),  
            ("dy", ctypes.c_long),  
            ("mouseData", ctypes.c_ulong),  
            ("dwFlags", ctypes.c_ulong),  
            ("time", ctypes.c_ulong),  
            ("dwExtraInfo", PUL)]
```

class Input_I(ctypes.Union):

```
_fields_ = [("ki", KeyBdInput),  
            ("mi", MouseInput),
```

```
("hi", HardwareInput)]
```

```
class Input(ctypes.Structure):
```

```
    _fields_ = [("type", ctypes.c_ulong),  
                ("ii", Input_I)]
```

```
# Actuals Functions
```

```
def PressKey(hexKeyCode):
```

```
    extra = ctypes.c_ulong(0)  
    ii_ = Input_I()  
    ii_.ki = KeyBdInput( 0, hexKeyCode, 0x0008, 0, ctypes.pointer(extra) )  
    x = Input( ctypes.c_ulong(1), ii_ )  
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))
```

```
def ReleaseKey(hexKeyCode):
```

```
    extra = ctypes.c_ulong(0)  
    ii_ = Input_I()  
    ii_.ki = KeyBdInput( 0, hexKeyCode, 0x0008 | 0x0002, 0, ctypes.pointer(extra) )  
    x = Input( ctypes.c_ulong(1), ii_ )  
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x), ctypes.sizeof(x))
```

```
if __name__ == '__main__':
```

```
    PressKey(0x11)  
    time.sleep(1)  
    ReleaseKey(0x11)
```

```
time.sleep(1)
```

File: steering.py

Documented code for steering.py:

```
import cv2
```

```
import imutils
```

```
from imutils.video import VideoStream
```

```
import numpy as np
```

```
from directkeys import A,D,Space,ReleaseKey ,PressKey
```

```
cam = VideoStream(src = 0).start()
```

```
currentKey=list()
```

```
while True :
```

```
    key = False
```

```
    img = cam.read()
```

```
    img = np.flip(img , axis=1)
```

```
    img = np.array(img)
```

```
    hsv= cv2.cvtColor(img ,cv2.COLOR_BGR2HSV)
```

```
    blurred = cv2.GaussianBlur(hsv ,(11,11),0)
```

```
    colorLower = np.array([31,33 ,153])
```

```
    colorUpper = np.array([100,255,255])
```

```
mask = cv2.inRange(blurred,colorLower , colorUpper)
```

```
mask = cv2.morphologyEx(mask , cv2.MORPH_OPEN , np.ones((5,5),np.uint8))
```

```
mask = cv2.morphologyEx(mask , cv2.MORPH_CLOSE , np.ones((5,5),np.uint8))
```

```
width = img.shape[1]
```

```
height = img.shape[0]
```

```
upContour = mask[0:height//2, 0:width]
```

```
downContour = mask[3*height//4:height, 2*width//5:3*width//5 ]
```

```
cnts_up = cv2.findContours(upContour, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
cnts_up = imutils.grab_contours(cnts_up)
```

```
cnts_down = cv2.findContours(downContour, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
cnts_down = imutils.grab_contours(cnts_down)
```

```
if len(cnts_up) > 0:
```

```
    c = max(cnts_up ,key=cv2.contourArea)
```

```
    M = cv2.moments(c)
```

```
    cX = int(M["m10"]/M["m00"])
```

```
    if cX < (width//2 -35):
```

```
        PressKey(A)
```

```
        key = True
```

```
        currentKey.append(A)
```

```
    elif cX > (width//2 +35):
```

```
PressKey(D)
```

```
key = True
```

```
currentKey.append(D)
```

```
if len(cnts_down) > 0:
```

```
    PressKey(Space)
```

```
    key = True
```

```
    currentKey.append(Space)
```

```
img = cv2.rectangle(img , (0,0),(width//2-35,height//2),(0,255,0),1)
```

```
cv2.putText(img ,"LEFT",(110,30),cv2.FONT_HERSHEY_COMPLEX,1,(139,0,0))
```

```
img = cv2.rectangle(img , (width//2+35,0),(width, height//2),(0,255,0),1)
```

```
cv2.putText(img ,"RIGHT",(440,30),cv2.FONT_HERSHEY_COMPLEX,1,(139,0,0))
```

```
img = cv2.rectangle(img , (2*(width//5),3*height//4),(3*width//5,height),(0,255,0),1)
```

```
cv2.putText(img ,"NITRO",(2*(width//5)+20,height-10),cv2.FONT_HERSHEY_COMPLEX,1,(139,0,0))
```

```
cv2.imshow("Steering Whele [HAWK-AI]", img)
```

```
if not key and len(currentKey)!=0:
```

```
    for current in currentKey:
```

```
        ReleaseKey(current)
```

```
Key = cv2.waitKey(1) & 0xFF
```

```
if Key == ord('q'):
```

```
    break
```

```
cv2.destroyAllWindows()
```

Folder: People Counter

Folder: __pycache__

File: centroidtracker.py

Documented code for centroidtracker.py:

```
# import the necessary packages
```

```
from scipy.spatial import distance as dist
```

```
from collections import OrderedDict
```

```
import numpy as np
```

```
class CentroidTracker:
```

```
    def __init__(self, maxDisappeared=50, maxDistance=50):
```

```
        # initialize the next unique object ID along with two ordered
```

```
        # dictionaries used to keep track of mapping a given object
```

```
        # ID to its centroid and number of consecutive frames it has
```

```
        # been marked as "disappeared", respectively
```

```
        self.nextObjectID = 0
```

```
        self.objects = OrderedDict()
```

```
        self.disappeared = OrderedDict()
```

```
        self.bbox = OrderedDict() # CHANGE
```



```
# store the number of maximum consecutive frames a given
# object is allowed to be marked as "disappeared" until we
# need to deregister the object from tracking
self.maxDisappeared = maxDisappeared
```

```
# store the maximum distance between centroids to associate
# an object -- if the distance is larger than this maximum
# distance we'll start to mark the object as "disappeared"
self.maxDistance = maxDistance
```

```
def register(self, centroid, inputRect):
```

```
# when registering an object we use the next available object
# ID to store the centroid
self.objects[self.nextObjectID] = centroid
self.bbox[self.nextObjectID] = inputRect # CHANGE
self.disappeared[self.nextObjectID] = 0
self.nextObjectID += 1
```

```
def deregister(self, objectID):
```

```
# to deregister an object ID we delete the object ID from
# both of our respective dictionaries
del self.objects[objectID]
del self.disappeared[objectID]
del self.bbox[objectID] # CHANGE
```

```

def update(self, rects):

    # check to see if the list of input bounding box rectangles
    # is empty

    if len(rects) == 0:

        # loop over any existing tracked objects and mark them
        # as disappeared

        for objectID in list(self.disappeared.keys()):

            self.disappeared[objectID] += 1

            # if we have reached a maximum number of consecutive
            # frames where a given object has been marked as
            # missing, deregister it

            if self.disappeared[objectID] > self.maxDisappeared:

                self.deregister(objectID)

        # return early as there are no centroids or tracking info
        # to update

        # return self.objects

        return self.bbox

    # initialize an array of input centroids for the current frame

    inputCentroids = np.zeros((len(rects), 2), dtype="int")

    inputRects = []

    # loop over the bounding box rectangles

    for (i, (startX, startY, endX, endY)) in enumerate(rects):

        # use the bounding box coordinates to derive the centroid

```

```
cX = int((startX + endX) / 2.0)
```

```
cY = int((startY + endY) / 2.0)
```

```
inputCentroids[i] = (cX, cY)
```

```
inputRects.append(rects[i]) # CHANGE
```

```
# if we are currently not tracking any objects take the input
```

```
# centroids and register each of them
```

```
if len(self.objects) == 0:
```

```
    for i in range(0, len(inputCentroids)):
```

```
        self.register(inputCentroids[i], inputRects[i]) # CHANGE
```

```
# otherwise, are are currently tracking objects so we need to
```

```
# try to match the input centroids to existing object
```

```
# centroids
```

```
else:
```

```
    # grab the set of object IDs and corresponding centroids
```

```
    objectIDs = list(self.objects.keys())
```

```
    objectCentroids = list(self.objects.values())
```

```
# compute the distance between each pair of object
```

```
# centroids and input centroids, respectively -- our
```

```
# goal will be to match an input centroid to an existing
```

```
# object centroid
```

```
D = dist.cdist(np.array(objectCentroids), inputCentroids)
```

```
# in order to perform this matching we must (1) find the
```

```

# smallest value in each row and then (2) sort the row
# indexes based on their minimum values so that the row
# with the smallest value as at the *front* of the index
# list
rows = D.min(axis=1).argsort()

# next, we perform a similar process on the columns by
# finding the smallest value in each column and then
# sorting using the previously computed row index list
cols = D.argmin(axis=1)[rows]

# in order to determine if we need to update, register,
# or deregister an object we need to keep track of which
# of the rows and column indexes we have already examined
usedRows = set()
usedCols = set()

# loop over the combination of the (row, column) index
# tuples
for (row, col) in zip(rows, cols):
    # if we have already examined either the row or
    # column value before, ignore it
    if row in usedRows or col in usedCols:
        continue

    # if the distance between centroids is greater than

```

```

# the maximum distance, do not associate the two
# centroids to the same object
if D[row, col] > self.maxDistance:
    continue

# otherwise, grab the object ID for the current row,
# set its new centroid, and reset the disappeared
# counter
objectID = objectIDs[row]
self.objects[objectID] = inputCentroids[col]
self.bbox[objectID] = inputRects[col] # CHANGE
self.disappeared[objectID] = 0

# indicate that we have examined each of the row and
# column indexes, respectively
usedRows.add(row)
usedCols.add(col)

# compute both the row and column index we have NOT yet
# examined
unusedRows = set(range(0, D.shape[0])).difference(usedRows)
unusedCols = set(range(0, D.shape[1])).difference(usedCols)

# in the event that the number of object centroids is
# equal or greater than the number of input centroids
# we need to check and see if some of these objects have

```

```

# potentially disappeared

if D.shape[0] >= D.shape[1]:

    # loop over the unused row indexes

    for row in unusedRows:

        # grab the object ID for the corresponding row

        # index and increment the disappeared counter

        objectID = objectIDs[row]

        self.disappeared[objectID] += 1


        # check to see if the number of consecutive

        # frames the object has been marked "disappeared"

        # for warrants deregistering the object

        if self.disappeared[objectID] > self.maxDisappeared:

            self.deregister(objectID)


# otherwise, if the number of input centroids is greater

# than the number of existing object centroids we need to

# register each new input centroid as a trackable object

else:

    for col in unusedCols:

        self.register(inputCentroids[col], inputRects[col])


# return the set of trackable objects

# return self.objects

return self.bbox

```

File: distance.py

Documented code for distance.py:

```
import cv2
```

```
import datetime
```

```
import imutils
```

```
import numpy as np
```

```
from centroidtracker import CentroidTracker
```

```
from itertools import combinations
```

```
import math
```

```
protopath="MobileNetSSD_deploy.prototxt"
```

```
modelpath="MobileNetSSD_deploy.caffemodel"
```

```
detector=cv2.dnn.readNetFromCaffe(protopath,caffeModel=modelpath)
```

```
tracker=CentroidTracker(maxDisappeared=80,maxDistance=90)
```

```
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",  
            "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",  
            "dog", "horse", "motorbike", "person", "pottedplant", "sheep",  
            "sofa", "train", "tvmonitor"]
```

```
def non_max_supression_fast(boxes,overlapThresh):
```

```
    try:
```

```
        if len(boxes)==0:
```

```

    return[]

if boxes.dtype.kind=="i":
    boxes=boxes.astype("float")

pick=[]
x1=boxes[:,0]
y1=boxes[:,1]
x2=boxes[:,2]
y2=boxes[:,3]

area=(x2-x1+1)*(y2-y1+1)
idxs=np.argsort(y2)

while len(idxs)>0:
    last=len(idxs)-1
    i=idxs[last]
    pick.append(i)
    xx1=np.maximum(x1[i],x1[idxs[:last]])
    yy1=np.maximum(y1[i],y1[idxs[:last]])
    xx2=np.minimum(x2[i],y2[idxs[:last]])
    yy2=np.minimum(y2[i],y2[idxs[:last]])

    w=np.maximum(0,xx2-xx1+1)
    h=np.maximum(0,yy2-yy1+1)

    overlap=(w*h)/area[idxs[:last]]

```



```
idxs=np.delete(idxs,np.concatenate([[last],
                                     np.where(overlap>overlapThresh)[0]]))
```

```
return boxes[pick].astype("int")
```

```
except Exception as e:
```

```
print("Exception occurred in non_max_suppression:{}".format(e))
```

```
def main () :
```

```
cap=cv2.VideoCapture("videos/testvideo2.mp4")
```

```
fps_start=datetime.datetime.now()
```

```
fps=0
```

```
total_frames=0
```

```
centroid_dict=dict()
```

```
while True:
```

```
ret, frame= cap.read()
```

```
frame=imutils.resize(frame,width=600)
```

```
total_frames=total_frames+1
```

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

```
blob=cv2.dnn.blobFromImage(frame,0.007843,(W,H),127.5)
```

```
detector.setInput(blob)
```

```
person_detections=detector.forward()
```

```
rects=[]
```

```
for i in np.arange(0,person_detections.shape[2]):
```

```
    confidence=person_detections[0,0,i,2]
```

```
    if confidence>0.5:
```

```
        idx=int(person_detections[0,0,i,1])
```

```
        if CLASSES[idx] != "person":
```

```
            continue
```

```
        person_box= person_detections[0,0,i,3:7] * np.array([W,H,W,H])
```

```
        (startX,startY,endX,endY)=person_box.astype("int")
```

```
        rects.append(person_box)
```

```
boundingboxes=np.array(rects)
```

```
boundingboxes=boundingboxes.astype(int)
```

```
rects=non_max_supression_fast(boundingboxes,0.3)
```

```
objects = tracker.update(rects)
```

```
for (objectId,bbox) in objects.items():
```

```
    x1,y1,x2,y2=bbox
```

```
    x1=int(x1)
```

```
    y1=int(y1)
```

```
    x2=int(x2)
```

```
y2=int(y2)
```

```
cX=int((x1+x2)/2.0)
```

```
cY=int((y1+y2)/2.0)
```

```
centroid_dict[objectId]=(cX,cY,x1,y1,x2,y2)
```

```
#text="ID:{}".format(objectId)
```

```
#cv2.putText(frame,text,(x1,y1-5),cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),1)
```

```
red_zone_list=[]
```

```
for (id1,p1),(id2,p2) in combinations(centroid_dict.items(),2):
```

```
    dx,dy=p1[0]-p2[0],p1[1]-p2[1]
```

```
    distance=math.sqrt(dx*dx+dy*dy)
```

```
    if distance<75.0:
```

```
        if id1 not in red_zone_list:
```

```
            red_zone_list.append(id1)
```

```
        if id2 not in red_zone_list:
```

```
            red_zone_list.append(id2)
```

```
for id,box in centroid_dict.items():
```

```
    if id in red_zone_list:
```

```
        cv2.rectangle(frame,(box[2],box[3]),(box[4],box[5]),(0,0,255),2)
```

```
    else:
```

```
        cv2.rectangle(frame,(box[2],box[3]),(box[4],box[5]),(0,255,0),2)
```

```
fps_and_time=datetime.datetime.now()

time_diff=fps_and_time - fps_start

if time_diff.seconds==0:

    fps=0.0

else:

    fps=(total_frames/time_diff.seconds)


fps_text="FPS:{:.2f}".format(fps)


cv2.putText(frame,fps_text,(5,30),cv2.FONT_HERSHEY_COMPLEX,1,(0,0,255),2)

cv2.imshow("FPS",frame)

key=cv2.waitKey(1)

if key==ord('q'):

    break

cv2.destroyAllWindows()

main()
```

File: final.py

Documented code for final.py:

```
import cv2

import datetime

import imutils

import numpy as np

from centroidtracker import CentroidTracker

import psycopg2

import psycopg2.extras
```

```
#initialize database connection
```

```
hostname = 'localhost'
```

```
database='postgres'
```

```
username='postgres'
```

```
pwd='password'
```

```
port_id=5432
```

```
conn=None
```

```
cur=None
```

```
#importing module files
```

```
protopath="MobileNetSSD_deploy.prototxt"
```

```
modelpath="MobileNetSSD_deploy.caffemodel"
```

```
detector=cv2.dnn.readNetFromCaffe(prototxt=protopath,caffeModel=modelpath)
```

```
tracker=CentroidTracker(maxDisappeared=80,maxDistance=90)
```

```
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",  
            "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",  
            "dog", "horse", "motorbike", "person", "pottedplant", "sheep",  
            "sofa", "train", "tvmonitor"]
```

#function for generating unique ID's

```
def non_max_supression_fast(boxes,overlapThresh):
```

```
    try:
```

```
        if len(boxes)==0:
```

```
            return[]
```

```
        if boxes.dtype.kind=="i":
```

```
            boxes=boxes.astype("float")
```

```
        pick=[]
```

```
        x1=boxes[:,0]
```

```
        y1=boxes[:,1]
```

```
        x2=boxes[:,2]
```

```
        y2=boxes[:,3]
```

```
        area=(x2-x1+1)*(y2-y1+1)
```

```
        idxs=np.argsort(y2)
```

```
        while len(idxs)>0:
```

```
            last=len(idxs)-1
```

```
            i=idxs[last]
```

```
            pick.append(i)
```

```
            xx1=np.maximum(x1[i],x1[idxs[:last]])
```

```
            yy1=np.maximum(y1[i],y1[idxs[:last]])
```

```
            xx2=np.minimum(x2[i],y2[idxs[:last]])
```

```
yy2=np.minimum(y2[i],y2[idxs[:last]])
```

```
w=np.maximum(0,xx2-xx1+1)
```

```
h=np.maximum(0,yy2-yy1+1)
```

```
overlap=(w*h)/area[idxs[:last]]
```

```
idxs=np.delete(idxs,np.concatenate([[last],  
                                     np.where(overlap>overlapThresh)[0])))
```

```
return boxes[pick].astype("int")
```

```
except Exception as e:
```

```
print("Exception occurred in non_max_suppression:{}".format(e))
```

```
def main() :
```

```
cap=cv2.VideoCapture(0) #give any video input or use cap=cv2.VideoCapture(0) to detect using webcam
```

```
fps_start=datetime.datetime.now()
```

```
fps=0
```

```
total_frames=0
```

```
lpc_count=0
```

```
opc_count=0
```

```
object_id_list=[]
```

```
while True:
```

```
ret, frame= cap.read()
```

```
frame=imutils.resize(frame,width=600)
```

```
total_frames=total_frames+1
```

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

```
blob=cv2.dnn.blobFromImage(frame,0.007843,(W,H),127.5)
```

```
detector.setInput(blob)
```

```
person_detections=detector.forward()
```

```
rects=[]
```

```
for i in np.arange(0,person_detections.shape[2]):
```

```
    confidence=person_detections[0,0,i,2]
```

```
    if confidence>0.5:
```

```
        idx=int(person_detections[0,0,i,1])
```

```
        if CLASSES[idx] != "person":
```

```
            continue
```

```
        person_box= person_detections[0,0,i,3:7] * np.array([W,H,W,H])
```

```
        (startX,startY,endX,endY)=person_box.astype("int")
```

```
        rects.append(person_box)
```

```
boundingboxes=np.array(rects)
```

```
boundingboxes=boundingboxes.astype(int)
```

```
rects=non_max_supression_fast(boundingboxes,0.3)
```



```
objects = tracker.update(rects)

for (objectId,bbox) in objects.items():

    x1,y1,x2,y2=bbox

    x1=int(x1)
    y1=int(y1)
    x2=int(x2)
    y2=int(y2)


    cv2.rectangle(frame,(x1,y1),(x2,y2),(0,255,0),2)

    text="ID:{}".format(objectId)

    cv2.putText(frame,text,(x1,y1-5),cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),1)


    if objectId not in object_id_list:

        object_id_list.append(objectId)


#displaying overall and live person count


lpc_count=len(objects)

opc_count=len(object_id_list)

lpc_txt="LIVE COUNT: {}".format(lpc_count)

opc_txt="TOTAL COUNT:{}".format(opc_count)


#Send Live and total count data to Database realtime
```

try:

```
with psycopg2.connect(  
    host=hostname,  
    dbname=database,  
    user=username,  
    password=pwd,  
    port=port_id) as conn:
```

```
with conn.cursor(cursor_factory=psycopg2.extras.DictCursor) as cur:
```

```
    insert_script='INSERT INTO analytics (count,id) VALUES (%s, %s)'
```

```
    insert_values=[(opc_txt,objectId)]
```

```
    for record in insert_values:
```

```
        cur.execute(insert_script,record)
```

```
    cur.execute('SELECT * FROM analytics')
```

```
    conn.commit()
```

```
except Exception as error:
```

```
    print(error)
```

finally:

if conn is not None:

conn.close()

cv2.putText(frame,lpc_txt,(5,60),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)

cv2.putText(frame,opc_txt,(5,90),cv2.FONT_HERSHEY_SIMPLEX,1,(0,0,255),2)

cv2.imshow("Frame",frame)

key=cv2.waitKey(1)

if key==ord('q'):

break

cv2.destroyAllWindows()

main()

Folder: Vehicle Count

File: vehicle.py

Documented code for vehicle.py:

import cv2

import numpy as np

import psycopg2

#Web Camera

cap=cv2.VideoCapture('video.mp4')

```
min_width_rect = 80 #minimum width of rectangle  
min_height_rect = 80 #minimum height of rectangle  
count_line_position=550
```

```
#Initialize Substructor
```

```
algo=cv2.bgsegm.createBackgroundSubtractorMOG()
```

```
def center(x,y,w,h):
```

```
    x1=int(w/2)
```

```
    y1=int(h/2)
```

```
    cx = x+x1
```

```
    cy = y+y1
```

```
    return cx,cy
```

```
detect = []
```

```
offset= 6 #allowing error between pixel
```

```
counter=0
```

```
while True:
```

```
    ret,frame1=cap.read()
```

```
    grey = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
```

```
    blur = cv2.GaussianBlur(grey,(3,3),5)
```

```
    #applying on each fram
```

```

img_sub = algo.apply(blur)

dilat = cv2.dilate(img_sub, np.ones((5,5)))

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))

dilate=cv2.morphologyEx(dilat,cv2.MORPH_CLOSE,kernel)

dilate=cv2.morphologyEx(dilate,cv2.MORPH_CLOSE,kernel)

contour,h=cv2.findContours(dilate,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)


cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(255,255,255),3)


for (i,c) in enumerate(contour):

    (x,y,w,h)=cv2.boundingRect(c)

    validate_counter=(w>=min_width_rect) and (h>=min_height_rect)

    if not validate_counter:

        continue


    cv2.rectangle(frame1,(x,y),(x+w,y+h),(0,255,0),2)

    cv2.putText(frame1,"Vehicle"+str(counter),(x,y-20),cv2.FONT_HERSHEY_COMPLEX,1,(255,0,0),2)


    counter1=center(x,y,w,h)

    detect.append(counter1)

    cv2.circle(frame1,counter1,4,(0,0,255),-1)


for (x,y) in detect:

    if y<(count_line_position+offset) and y>(count_line_position-offset):

        counter+=1

```

```
cv2.line(frame1,(25,count_line_position),(1200,count_line_position),(0,0,255),3)

detect.remove((x,y))

count=str(counter)

print("Vehicle Counting:"+str(counter))
```

```
# establish a connection to the database
```

```
conn = psycopg2.connect(

    host= 'localhost',

    database='postgres',

    user='postgres',

    password='yourpassword',

)
```

```
# create a cursor object
```

```
cur = conn.cursor()
```

```
# insert data into the table
```

```
cur.execute("INSERT INTO vehiclecounter (VehicleCount) VALUES (%s)", (count,))
```

```
# commit the changes
```

```
conn.commit()
```

```
# close the cursor and connection
```

```
cur.close()
```

```
conn.close()
```

```
cv2.putText(frame1, "Vehicle Counter:"+str(counter),(450,70),cv2.FONT_HERSHEY_COMPLEX,2,(0,0,1
```

```
#cv2.imshow("Detector",dilate)
```

```
cv2.imshow("frame",frame1)
```

```
if cv2.waitKey(0) ==13:
```

```
    break
```

```
cap.release()
```

```
cv2.destroyAllWindows()
```

Folder: Virtual Keyboard

File: main.py

Documented code for main.py:

```
import cvzone
```

```
import cv2
```

```
from cvzone.HandTrackingModule import HandDetector
```

```
from time import sleep
```

```
import numpy as np
```

```
from pynput.keyboard import Controller , Key
```

```
cap= cv2.VideoCapture(0)
```

```
#Window size
```

```
cap.set(3,1280)
```

```
cap.set(4,720)
```

```
detector = HandDetector(detectionCon=0.8 , maxHands=2)
```

```
keys = [["Q", "W", "E", "R", "T", "Y","U", "I", "O", "P"],
```

```
        ["A", "S", "D", "F", "G", "H", "J", "K", "L", ";,"],
```

```
        ["Z","X","C","V","B", "N ", "M"]]
```

```
finaltext = ""
```

```
keyboard = Controller()
```

```
#Handles Drawing of all the buttons :
```

```
#Instead of writing one by one button make a list
```

```
def drawALL( img , buttonList):
```

```
    imgNew = np.zeros_like(img,np.uint8)
```

```
    for button in buttonList:
```

```
        x,y = button.pos
```

```
        cvzone.cornerRect(imgNew,(button.pos[0], button.pos[1],button.size[0], button.size[1]),20 ,rt=0)
```

```
        cv2.rectangle(imgNew ,button.pos,(x + button.size[0], y + button.size[1]),(155,15,0), cv2.FILLED )
```

```
        cv2.putText(imgNew, button.text, (x+40,
```

```
                    y+60), cv2.FONT_HERSHEY_PLAIN,2, (255,255,255),3)
```

```
    out = img.copy()
```

```
    alpha = -0.5
```



```
mask = imgNew.astype(bool)
```

```
out[mask] = cv2.addWeighted(img , alpha, imgNew , 1 - alpha , 0)[mask]
```

```
return out
```

```
class Button():
```

```
    def __init__(self , pos , text , size=[85,85]):
```

```
        self.pos=pos
```

```
        self.size=size
```

```
        self.text=text
```

```
buttonList = []
```

```
for i in range(len(keys)):
```

```
    for j,key in enumerate(keys[i]):
```

```
        buttonList.append(Button([100 * j + 50 , 100*i+50],key))
```

```
while True :
```

```
    success, img = cap.read()
```

```
    #Find hands
```

```
    hands, img = detector.findHands(img)
```

```
bboxInfo= detector.findHands(img , draw=False)
```

```
lmList= detector.findHands(img , draw=False)
```

```
img = drawALL(img,buttonList)
```

```
#Check whether hand or not:
```

```
if hands:
```

```
    # Hand 1
```

```
    hand = hands[0]
```

```
    lmList = hand["lmList"] # List of 21 Landmarks points
```

```
    bbox = hand["bbox"] # Bounding Box info x,y,w,h
```

```
    centerPoint = hand["center"] # center of the hand cx,cy
```

```
    handType = hand["type"] # Hand Type Left or Right
```

```
if len(hands) == 2:
```

```
    hand2 = hands[1]
```

```
    lmList2 = hand2["lmList"] # List of 21 Landmarks points
```

```
    bbox2 = hand2["bbox"] # Bounding Box info x,y,w,h
```

```
    centerPoint2 = hand2["center"] # center of the hand cx,cy
```

```
    handType2 = hand2["type"] # Hand Type Left or Right
```

```
if lmList:
```

```
    for button in buttonList:
```

```
        x,y = button.pos
```

```
w,h = button.size
```

```
#Tip of finger is point no.8(index finger)
```

```
if x < lmList[8][0] < x+w and y<lmList[8][1]<y+h:
```

```
    cv2.rectangle(img ,button.pos,(x + w, y + h),(175,0,175), cv2.FILLED )
```

```
    cv2.putText(img , button.text, (x+20,y+65), cv2.FONT_HERSHEY_PLAIN,4, (255,255,255),
```

```
    l, _ , _= detector.findDistance(centerPoint,centerPoint2, img)
```

```
if l<1000:
```

```
    # keyboard.press(button.text)
```

```
    keyboard.press(button.text.lower() if len(button.text)==1 else Key.space) # Press lowercase
```

```
    cv2.rectangle(img ,button.pos,(x + w, y + h),(0,255,0), cv2.FILLED )
```

```
    cv2.putText(img , button.text, (x+20, y+65), cv2.FONT_HERSHEY_PLAIN,4, (255,255,255)
```

```
    finaltext += button.text
```

```
    sleep(1)
```

```
# ...
```

```
cv2.rectangle(img ,(50,350),(700,450),(0,0,0), cv2.FILLED )
```

```
cv2.putText(img , finaltext, (60,430), cv2.FONT_HERSHEY_PLAIN,4, (255,255,255),4)
```

```
cv2.imshow("image",img)
```

```
cv2.waitKey(1)==27
```