

**PES UNIVERSITY**

**(Established under Karnataka Act No. 16 of 2013)**

**100-ft Ring Road, Bengaluru-560 085, Karnataka, India**

***Report on***

**Traffic Light Control System**

**Using Image Processing**

***Submitted by***

**ROHAN N KALPAVRUKSHA (PES1201802830)**

**ROSHAN N KALPAVRUKSHA (PES1201802834)**

**SURIT (PES1201800589)**

***under the guidance of***

**Dr. Shikha Tripathi**

**Professor and Research Head**

**Department of Electronics and Communication Engineering**

**PES University**

**Bengaluru – 560085**

**INTRODUCTION**

In modern life we have to face with many problems one of which is traffic congestion becoming more serious day after day. It is said that the high tome of vehicles, the scanty infrastructure and the irrational distribution of the development are main reasons for augmented traffic jam.

Traffic has become a major concern in recent times due to the massive increase in the number of automobiles on the road. Expansion of roads and construction of flyovers too are unable to decrease the frequency of traffic jams at intersections. This calls for an effective traffic control and management system.

Today’s traffic management system has no emphasis on live traffic scenario, which leads to inefficient traffic management systems. This project has been implemented by using the python software and it aims to prevent heavy traffic congestion. Moreover, for implementing this project Image processing technique is used.

At first, film of a lane is captured by a camera. A web camera is placed in a traffic lane that will capture images of the road on which we want to control traffic. Then these images are efficiently processed to know the traffic density. According to the processed data from commands, the controller will send the command to the traffic LEDs to show particular time on the signal to manage traffic.

**THEORETICAL DETAILS**

Traffic congestion may result due to heavy traffic at a junction. To avoid congestion there are so many traffic management techniques available. But no technique is perfect by itself as the real time situations are generally continuously changing and the system has to adapt itself to change in the continuously changing circumstances. We have made an attempt to provide some traffic management strategy which is self-changing in nature, so as to fit into continuously changing real time traffic scenarios.

**Object Recognition** – The process of recognizing objects in videos and images is known as Object recognition. This computer vision technique enables the autonomous vehicles to classify and detect objects in real-time. There are many machine learning and deep learning algorithms for object detection and recognition, such as Support vector machine (SVM), Convolutional Neural Networks (CNNs), You Only Look Once (YOLO) model etc. We have made use of YOLO technique and COCO dataset.

YOLO is an extremely fast multi object detection algorithm which uses convolutional neural network (CNN) to detect and identify objects. Common Objects in Context **(COCO)** is a database that aims to enable future research for object detection, instance segmentation, image captioning, and person key points localization.

**METHODOLOGY**

**INPUT –** Image/video of vehicles on different lanes

**STEPS –**

* Calculates the no. of vehicles in the image/density of the picture frame of each lane
* Compares the values
* Sets longer duration of green light for a denser lane and a shorter duration of green light for a less dense lane.

**OUTPUT –** Number of vehicles in the frame.

Duration for which the green light is ‘ON’.

Recognizes the objects present in the frame (such as bicycle, car, motorbike, auto…).

**Traffic light signal used in our program**

|  |  |
| --- | --- |
| **Green light period** | **Vehicle traffic situation** |
| 10 seconds | 1 to 3 vehicles |
| 20 seconds | 3 to 5 vehicles |
| 30 seconds | 5 to 8 vehicles |

These values can be modified and updated as per needed in the real time situation.

**CODE**

import cv2 as cv        # Open cv

import numpy as np      # numpy - mathematical operations in python

# Object recognition Get the names of the output layers

def getOutputsNames(net):

    # Get the names of all the layers in the network

    layersNames = net.getLayerNames()

    # Get the names of the output layers, i.e. the layers with unconnected outputs

    return [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# Draw the predicted bounding box

def drawPred(classId, conf, left, top, right, bottom):

    # Draw a bounding box.

    cv.rectangle(frame, (left, top), (right, bottom), (0, 0, 255))

    label = '%.2f' % conf

    # Get the label for the class name and its confidence

    if classes:

        assert (classId < len(classes))

        label = '%s:%s' % (classes[classId], label)

    # Display the label at the top of the bounding box

    labelSize, baseLine = cv.getTextSize(label, cv.FONT\_HERSHEY\_SIMPLEX, 0.5, 1)

    top = max(top, labelSize[1])

    cv.putText(frame, label, (left, top), cv.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255))

# Remove the bounding boxes with low confidence using non-maxima suppression

def postprocess(frame, outs):

    frameHeight = frame.shape[0]

    frameWidth = frame.shape[1]

    global count

    classIds = []

    confidences = []

    boxes = []

    # Scan through all the bounding boxes output from the network and keep only the

    # ones with high confidence scores. Assign the box's class label as the class with the highest score.

    classIds = []

    confidences = []

    boxes = []

    for out in outs:

        for detection in out:

            scores = detection[5:]

            classId = np.argmax(scores)

            confidence = scores[classId]

            if confidence > confThreshold:

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

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

                width = int(detection[2] \* frameWidth)

                height = int(detection[3] \* frameHeight)

                left = int(center\_x - width / 2)

                top = int(center\_y - height / 2)

                classIds.append(classId)

                confidences.append(float(confidence))

                boxes.append([left, top, width, height])

    # Perform non maximum suppression to eliminate redundant overlapping boxes with

    # lower confidences.

    indices = cv.dnn.NMSBoxes(boxes, confidences, confThreshold, nmsThreshold)

    for i in indices:

        i = i[0]

        box = boxes[i]

        left = box[0]

        top = box[1]

        width = box[2]

        height = box[3]

        drawPred(classIds[i], confidences[i], left, top, left + width, top + height)

        if classIds[i] in [1, 2, 3, 5, 7]:  # 0

            count = count + 1

# Initialize the parameters

confThreshold = 0.5  # Confidence threshold

nmsThreshold = 0.4  # Non-maximum suppression threshold

inpWidth = 416  # Width of network's input image

inpHeight = 416  # Height of network's input image

# Load names of classes

classesFile = "coco.names"

classes = None

with open(classesFile, 'rt') as f:

    classes = f.read().rstrip('\n').split('\n')

# Give the configuration and weight files for the model and load the network using them.

modelConfiguration = "yolov3.cfg"

modelWeights = "yolov3.weights"

net = cv.dnn.readNetFromDarknet(modelConfiguration, modelWeights)

net.setPreferableBackend(cv.dnn.DNN\_BACKEND\_OPENCV)

net.setPreferableTarget(cv.dnn.DNN\_TARGET\_CPU)

# if using image then make the output file a jpeg file.

outputFile = "result.avi"

# Process inputs

# winName = 'DL OD with OpenCV'

# cv.namedWindow(winName, cv.WINDOW\_NORMAL)

# cv.resizeWindow(winName, 1000, 1000)

# if using an image, put the path of the image over here.

cap = cv.VideoCapture("video.avi")

vid\_writer = cv.VideoWriter(outputFile, cv.VideoWriter\_fourcc('M', 'J', 'P', 'G'), 10,

                            (round(cap.get(cv.CAP\_PROP\_FRAME\_WIDTH)), round(cap.get(cv.CAP\_PROP\_FRAME\_HEIGHT))))

while cap.isOpened():

    count = 0

    # get frame from video

    hasFrame, frame = cap.read()

    # Create a 4D blob from a frame

    blob = cv.dnn.blobFromImage(frame, 1 / 255, (inpWidth, inpHeight), [0, 0, 0], 1, crop=False)

    # Set the input to the net

    net.setInput(blob)

    outs = net.forward(getOutputsNames(net))

    postprocess(frame, outs)

    print("The number of vehicles on road are: " + str(count))

    # store the output image

    # cv.imwrite(outputFile, frame.astype(np.uint8))

    # show the output image

    # cv.imshow(winName, frame)

    # store the output video

    vid\_writer.write(frame.astype(np.uint8))

    # Sample logic

    # if count > 20:

    #   Set timer to max value

    # else if count > 10:

    #   Set timer to lower value

    # else:

    #   Set timer to least value

    if (1 < count & count <= 3):

       print ("Green on for 10 sec")

    elif (3 < count & count <= 5):

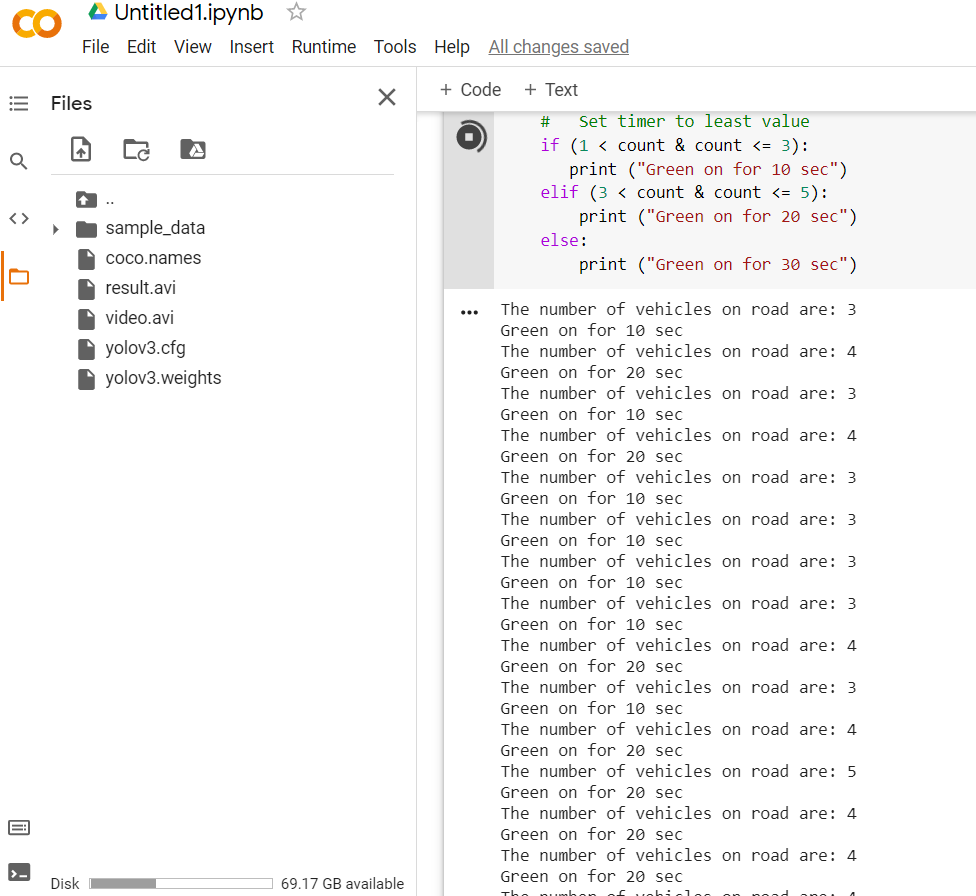
        print ("Green on for 20 sec")

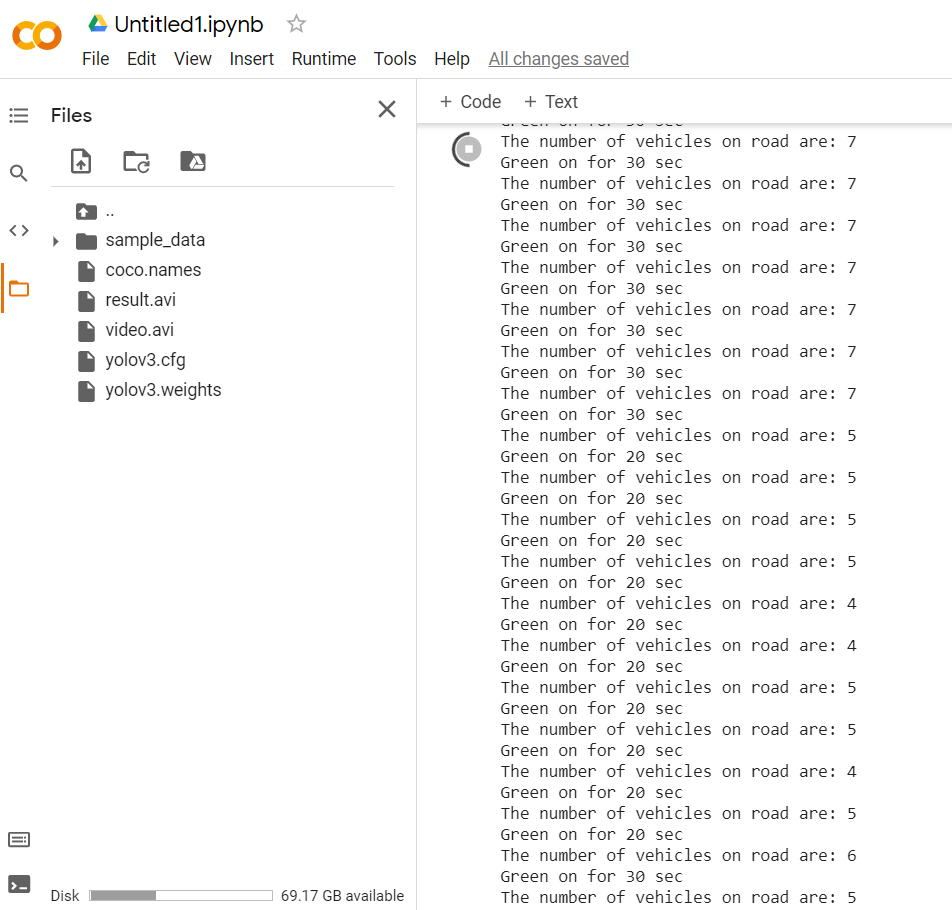
    else:

        print ("Green on for 30 sec")

**RESULT**

**Screenshots of number of vehicles and duration of green light displayed**





**Screenshots of object recognition in video**

****

In this project, a method for estimating the traffic using Image Processing is presented. This is done by using the camera images captured and videos taken are converted to the image sequences. Each image is processed separately and the number of has been counted and traffic is controlled. The advantages of this new method include benefits such as use of image processing over sensors, low cost, easy setup and relatively good accuracy and speed. Because this method has been implemented using Image Processing and Open cv software, production costs are low while achieving high speed and accuracy.