

## 1. INTRODUCTION

Today, many new technological developments have occurred. As a result of these technological developments, people may face several crucial problems. Some of the negativities to be experienced with the detection of such problems can be minimized through various approaches. Because of this; it is necessary to fall in parallel with technological and scientific developments, traffic accidents are still at the forefront people's daily life both in our country and also all over the world. Compared to air, sea and railway traffic, traffic on highways continues to be a significant problem in current life. Because no human being cannot be thought without driving out on a daily basis, traffic accident are considered to be an ordinary event of each human being. For this reason, it is essential for any person who involved in a traffic accident to objectively present the possible defect situations in a technical, legal and scientific way. So, vehicle detection and tracking systems gather lots of attention from the researcher's and is highly focused on by people. By detecting the number of cars in traffic prone area we can minimize the speed of car accordingly. This will minimize the traffic of that particular area and prevent the accidents. This project provide the family security by sending each data on their mobile phone.

## **1.1 Purpose:**

This project predicts the traffic prone area and accident prone area. By this project we provide security to the family. This project can minimize the traffic by detecting the number of cars on particular area and accordingly slow down the speed and this also minimize the accidents.

## **1.2 Scope:**

. In the near future, it will be used by many people before they went outside by cars.

.

## **1.3 Motivation:**

This projects help the users to take correct decision regarding the speed of the car in traffic prone area, depicts the no of cars near the users so that they can take valid action.

## 2. Software Requirement Analysis

The technologies and tools used by me to develop this machine are:

Library used is:

**Open CV:** Open CV was started at Intel in 1999 by **Gary Brodsky** and the first release came out in 2000. **VA dim Pisarevsky** joined Gary Brodsky to manage Intel's Russian software Open CV team. In 2005, Open CV was used on Stanley, the vehicle who won 2005 DARPA Grand Challenge. Later its active development continued under the support of Willow Garage, with Gary Brodsky and VA dim Pisarevsky leading the project. Right now, Open CV supports a lot of algorithms related to Computer Vision and Machine Learning and it is expanding day-by-day.

Open CV-Python is the Python API of Open CV. It combines the best qualities of Open CV C++ API and Python language.

**Open CV-Python:** Python is a general purpose programming language started by **Guido van Rossum**, which became very popular in short time mainly because of its simplicity and code readability. It enables the programmer to express his ideas in fewer lines of code without reducing any readability.

Compared to other languages like C/C++, Python is slower. But another important feature of Python is that it can be easily extended with C/C++. This feature helps us to write computationally intensive codes in C/C++ and create a Python wrapper for it so that we can use these wrappers as Python modules. This gives us two advantages: first, our code is as fast as original C/C++ code (since it is the actual C++ code working in background) and second, it is very easy to code in Python. This is how Open CV-Python works, it is a Python wrapper around original C++ implementation.

And the support of Numpy makes the task more easier. **Numpy** is a highly optimized library for numerical operations. It gives a MATLAB-style syntax. All the Open CV array structures are converted to-and-from Numpy arrays. So whatever operations you can do in Numpy, you can

combine it with Open CV, which increases number of weapons in your arsenal. Besides that, several other libraries like SciPy, Matplotlib which supports Numpy can be used with this.

So Open CV-Python is an appropriate tool for fast prototyping of computer vision problems.

**Open CV Needs You:** Since Open CV is an open source initiative, all are welcome to make contributions to this library. And it is same for this tutorial also.

So, if you find any mistake in this tutorial (whether it be a small spelling mistake or a big error in code or concepts, whatever), feel free to correct it.

And that will be a good task for fresher's who begin to contribute to open source projects. Just fork the Open CV in github, make necessary corrections and send a pull request to Open CV. Open CV developers will check your pull request, give you important feedback and once it passes the approval of the reviewer, it will be merged to Open CV. Then you become a open source contributor. Similar is the case with other tutorials, documentation etc.

As new modules are added to Open CV-Python, this tutorial will have to be expanded. So those who knows about particular algorithm can write up a tutorial which includes a basic theory of the algorithm and a code showing basic usage of the algorithm and submit it to Open CV.

Remember, we **together** can make this project a great success!!!

**Cascade Classifier:** Object Detection using Haar feature-based cascade classifiers is an effective object detection method proposed by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Here we will work with car detection. Initially, the algorithm needs a lot of positive images (images of cars) and negative images (images without cars) to train the classifier. Then we need to extract features from it.

A **Haar Cascade** is basically a **classifier** which is used to detect the object for which it has been trained for, from the source. The **Haar Cascade** is trained by superimposing the positive image over a set of negative images. The training is generally done on a server and on various stages.

## Object Detection Using Haar Cascade Classifier

**Abstract:** Object detection is an important feature of computer science. The benefits of object detection is however not limited to someone with a doctorate of informatics. Instead, object detection is growing deeper and deeper into the common parts of the information society, lending a helping hand wherever needed. This paper will address one such possibility, namely the help of a Haar-cascade classifier. The main focus will be on the case study of a vehicle detection and counting system and the possibilities it will provide in a semi-enclosed area - both the statistical kind and also for the common man. The goal of the system to be developed is to further ease and augment the everyday part of our lives.

**Introduction of Haar-like features:** A more sophisticated method is therefore required. One such method would be the detection of objects from images using features or specific structures of the object in question. However, there was a problem. Working with only image intensities, meaning the RGB pixel values in every single pixel in the image, made feature calculation rather computationally expensive and therefore slow on most platforms. This problem was addressed by the so called Haar-like features, developed by Viola and Jones on the basis of the proposal by Papa Georgiou et. al in 1998. A Haar-like feature considers neighboring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. An example of this would be the detection of human faces.

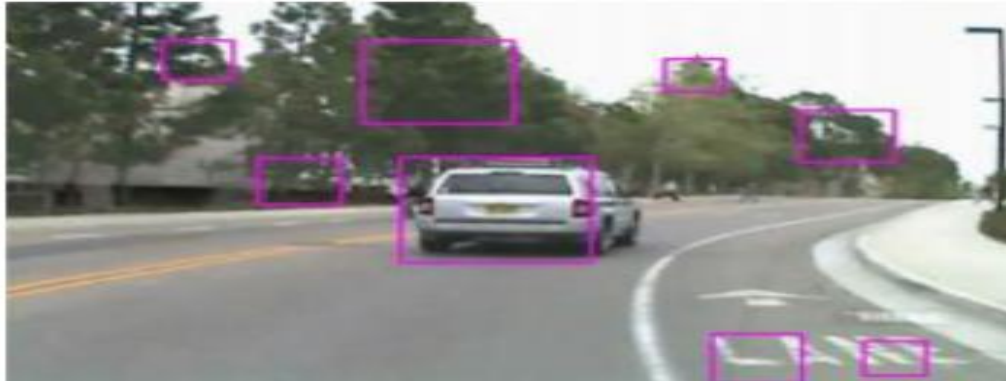
Commonly, the areas around the eyes are darker than the areas on the cheeks. One example of a Haar-like feature for face detection is therefore a set of two neighboring rectangular areas above the eye and cheek regions.

**Cascade classifier:** The cascade classifier consists of a list of stages, where each stage consists of a list of weak learners. The system detects objects in question by moving a window over the image. Each stage of the classifier labels the specific region defined by the current location of the window as either positive or negative – positive meaning that an object was found or negative means that the specified object was not found in the image. If the labelling yields a negative result, then the classification of this specific region is hereby complete and the location of the window is moved to the next location. If the labelling gives a positive result, then the region moves on to the next stage of classification. The classifier yields a final verdict of positive, when all the stages, including the last one, yield a result, saying that the object is found in the image. A true positive means that the object in question is indeed in the image and the classifier labels it as such – a positive result. A false positive means that the labelling process falsely determines, that the object is located in the image, although it is not. A false negative occurs when the classifier is unable to detect the actual object from the image and a true negative means that a non object was correctly classifier as not being the object in question. In order to work well, each stage of the cascade must have a low false negative rate, because if the actual object is classified as a non-object, then the classification of that branch stops, with no way to correct the mistake made. However, each stage can have a relatively high false positive rate, because even if the  $n$ th stage classifies the non-object as actually being the object, then this mistake can be fixed in  $n+1$ -th and subsequent stages of the classifier.

**Vehicle Detection and Tracking:** The system developed involves three main steps. Firstly, the preliminary car targets are generated with Haar-cascade classifier. Only the candidates that pass through all the stages are classified as positive and the ones that are rejected at any stage are classified as negative. The advantage is that the majority of the initial candidates are actually negative images, which usually cannot pass the first few stages. This early-stage rejection thus greatly reduces the overall computing time. After the first stage is complete, the target validation is used, which is based on car-light feature. Since there can be several false positives within the images output by the first stage of the system, which comes from the limitation of the training set. To reduce the false alarm rate, the algorithm uses the before mentioned car-light feature. Regardless of the shape, texture or color of the vehicle, they all share a common feature – they all have red lights in the rear. Thereby assuming that most of the false positives detected by the first stage do not possess such a feature, the result can thus be refined. Lastly, the results are further refined by Kalman tracking of the objects. The main idea of this step is based on a three-stage hypothesis tracking. Firstly, if a newly detected area appears and lasts for more than a certain number of steps, a hypothesis of the object is generated. Then it is predicted where the

next location of the object should be. If the object is not detected for a certain number of frames, the hypothesis is discarded. This method can thusly eliminate false positives that do not last long enough and still keep track of objects that are missing for only a short period in a detection step.

Below is the image of car detection using haar cascade classifier:



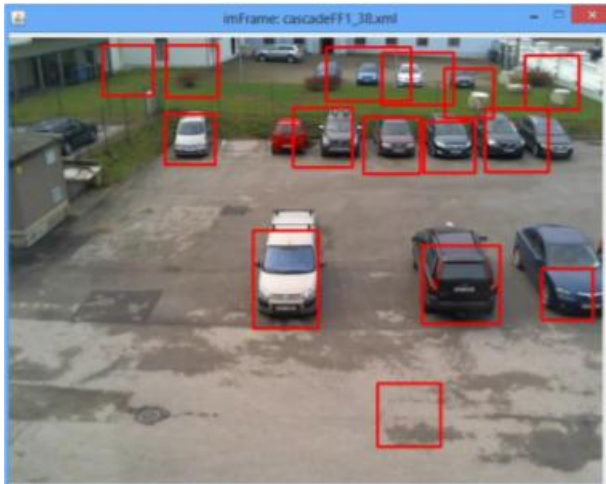
**Training cascade:** The training of the cascade proved to be no easy task. The first necessary bit was to gather the images, then create samples based on them and finally starting the training process. The open cv train cascade utility is an improvement over its predecessor in several aspects, one of them being that train cascade allows the training process to be multithreaded, which reduces the time it takes to finish the training of the classifier. This multithreaded approach is only applied during the pre calculation step however, so the overall time to train is still quite significant, resulting in hours, days and weeks of training time. [10, 11, 12] Since the training process needs a lot of positive and negative input images, which may not always be present, then a way to circumvent this is to use a tool for the creation of such positive images. Open CV built in mode allows to create more positive images with distorting the original positive image and applying a background image. However, it does not allow to do this for multiple images. By using the Perl script create samples to apply distortions in batch and the merge vec tool [11], it is possible to create such necessary files for each positive input file and then merging the outputted files together into one input file that Open CV can understand. Another important aspect to consider is the number of positives and negatives. When executing the command to start training, it is required to enter the number of positive and negative images process may determine that even the positive objects in the picture are actually negative ones, resulting in an empty result set. Fairly undefined behaviour can occur if the number of input images are too low, since the training program cannot get enough information on the actual object to be able to classify it correctly. One of the best results obtained in the course of this work is depicted on image 7. As one can observe, the classifier does detect some vehicles without any problems, but unfortunately also some areas of the pavement and some parts of grass are also classified as a car. Also some cars are not detected as standalone cars.

The time taken to train the classifier to detect at this level can be measured in days and weeks, rather than hours. Since the training process is fairly probabilistic, then a lot of work did also go

into testing the various parameters used in this work, from the number of input images to the subtle changes in the structuring element on the background removal, and verifying whether the output improved, decreased or remained unchanged. For the same reason, unfortunately the author of this work was unable to produce a proper classifier, which would give minimal false positives and maximal true positives.



**Depiction of the results of undertrained classifier**



**Best solution obtained**



**App Inventor:** MIT App Inventor is a web application integrated development environment originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT). It allows newcomers to computer programming to create application software(apps) for two operating systems (OS): Android (operating system)|Android, and iOS, which, as of 8 July 2019, is in final beta testing. It is free and open-source software released under Multi-licensing|dual licensing: a Creative Commons license#Attribution|Creative Commons Attribution ShareAlike 3.0 Unported license, and an [Apache License 2.0](#) for the [source code](#).

## User Implementations and Interface:

### CODE:

```
import cv2
#car_detetecion_harr_cascade.py
print(cv2.__version__)
import pyrebase

config = {
    'apiKey': "AIzaSyB577ujfKqZZJuTIW-SaaqggWdGQfZC3go",
    'authDomain': "project-323943198432",
    'databaseURL': "https://family-security-50c22.firebaseio.com/family_security",
    'projectId': "family-security-50c22",
    'storageBucket': "family-security-50c22",
}

firebase = pyrebase.initialize_app(config)

db = firebase.database()
cascade_src = 'C:\\Users\\sush1\\Downloads\\number_of_car_detection_harr_cascade-
master\\cars.xml'

# video_src = 'C:\\Users\\Naman\\Downloads\\Compressed\\vehicle_detection_haarcascades-
master\\vehicle_detection_haarcascades-master\\dataset\\video1.avi'
# video_src = "D:\\datasets\\example1.mp4"
video_src = 'C:\\Users\\sush1\\Downloads\\number_of_car_detection_harr_cascade-
master\\video2.avi'
font = cv2.FONT_HERSHEY_SIMPLEX
cap = cv2.VideoCapture(video_src)
car_cascade = cv2.CascadeClassifier(cascade_src)
total_cars = 0
flag = 19
a = -1
while True:
    flag += 1
    ret, img = cap.read()
    if (type(img) == type(None)):
        break
    # print(img.shape)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # img=cv2.resize(img,(400,400))
    cars = car_cascade.detectMultiScale(gray, 1.1, 1)
```

```

for (x, y, w, h) in cars:
    cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
    cv2.line(img, (0, 80), (300, 80), (255, 0, 0), 2)
    center_x = (x + (x + w)) // 2
    center_y = (y + (y + h)) // 2

    cv2.circle(img, (center_x, center_y), 2, (255, 0, 0), -1)
    if center_y == 80:
        total_cars += 1

cv2.putText(img, str(total_cars), (0, 50), font, 2, (0, 0, 255), 3, cv2.LINE_AA)

cv2.imshow('video', img)
if cv2.waitKey(33) == 27:
    break
if(total_cars==0):
    speed = 0
    print("NO Car is there around you ")
elif(1<=total_cars<=10):
    speed = 60
    print("Maintain your Speed 60 to 70 AND Drive Safely!")
elif(11<=total_cars<=20):
    speed = 50
    print("Maintain your Speed between 50 to 60 AND Drive Safely!")
elif(21<=total_cars<=30):
    speed = 30
    print("Maintain your Speed 30 AND Drive Safely!")
elif(31<=total_cars<=40):
    speed = 20
    print("Maintain your Speed 20 AND Drive Safely!")
else:
    speed = 40
    print("Maintain your Speed According to the Traffic,Maintain a particular distance
with other vehicles AND Drive Safely!")
print(total_cars)
if total_cars != a:
    db.child('total_cars').set(str(total_cars))
    db.child('speed').set(str(speed))
    a = total_cars

if cv2.waitKey(33) == 27:
    break

```

```
cv2.destroyAllWindows()
```

FIREBASE CODE:

```
import pyrebase
```

```
config = {  
    'apiKey': "AlzaSyDYr4tMw5XPCh_NBltDqQ4DyuvsI",  
    'authDomain': "eons-2c748.firebaseio.com",  
    'databaseURL': "https://eons-2c748.firebaseio.com",  
    'projectId': "eons-2c748",  
    'storageBucket': "eons-2c748.appspot.com",  
}
```

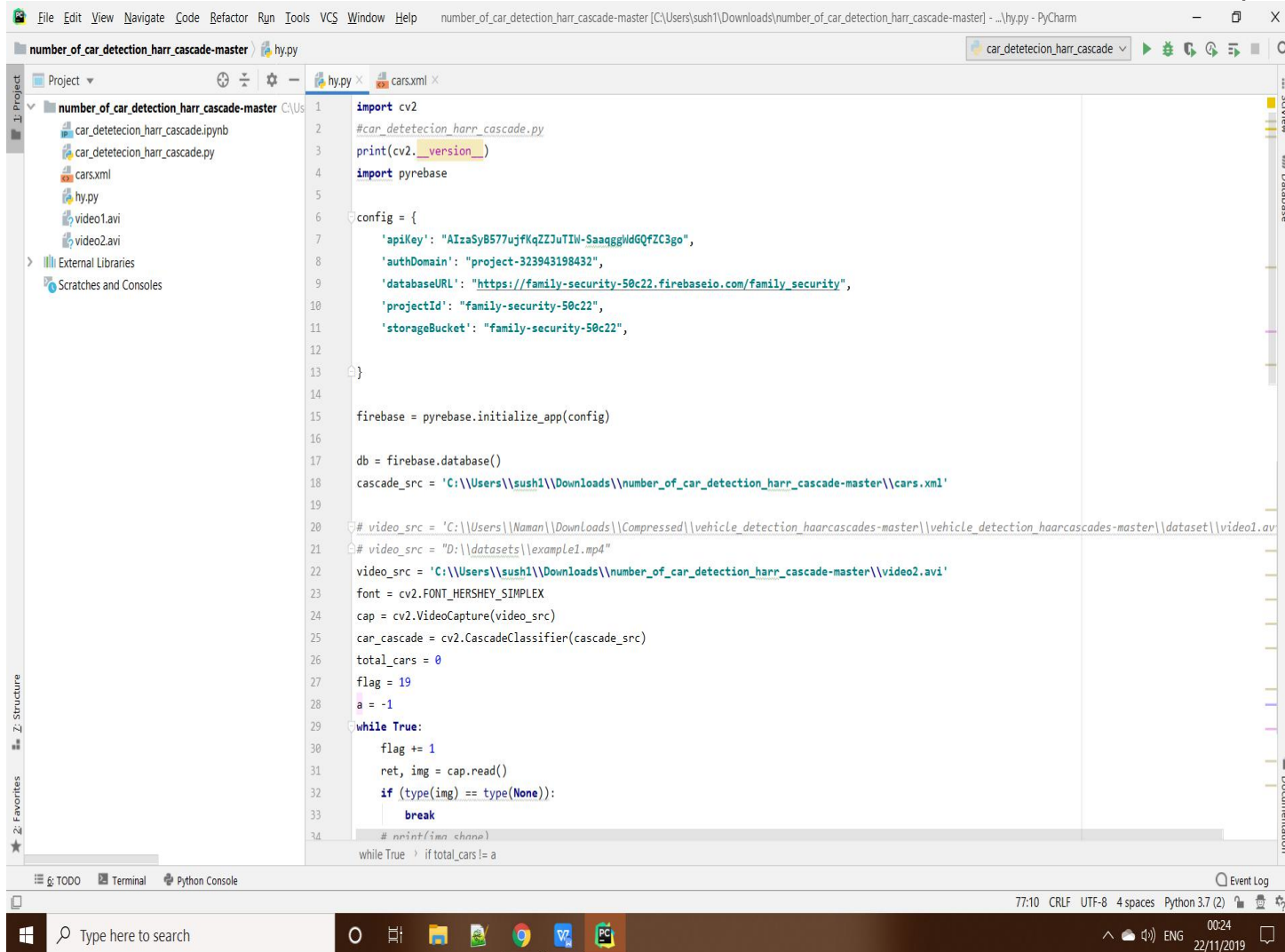
```
firebase = pyrebase.initialize_app(config)
```

```
db = firebase.database()
```

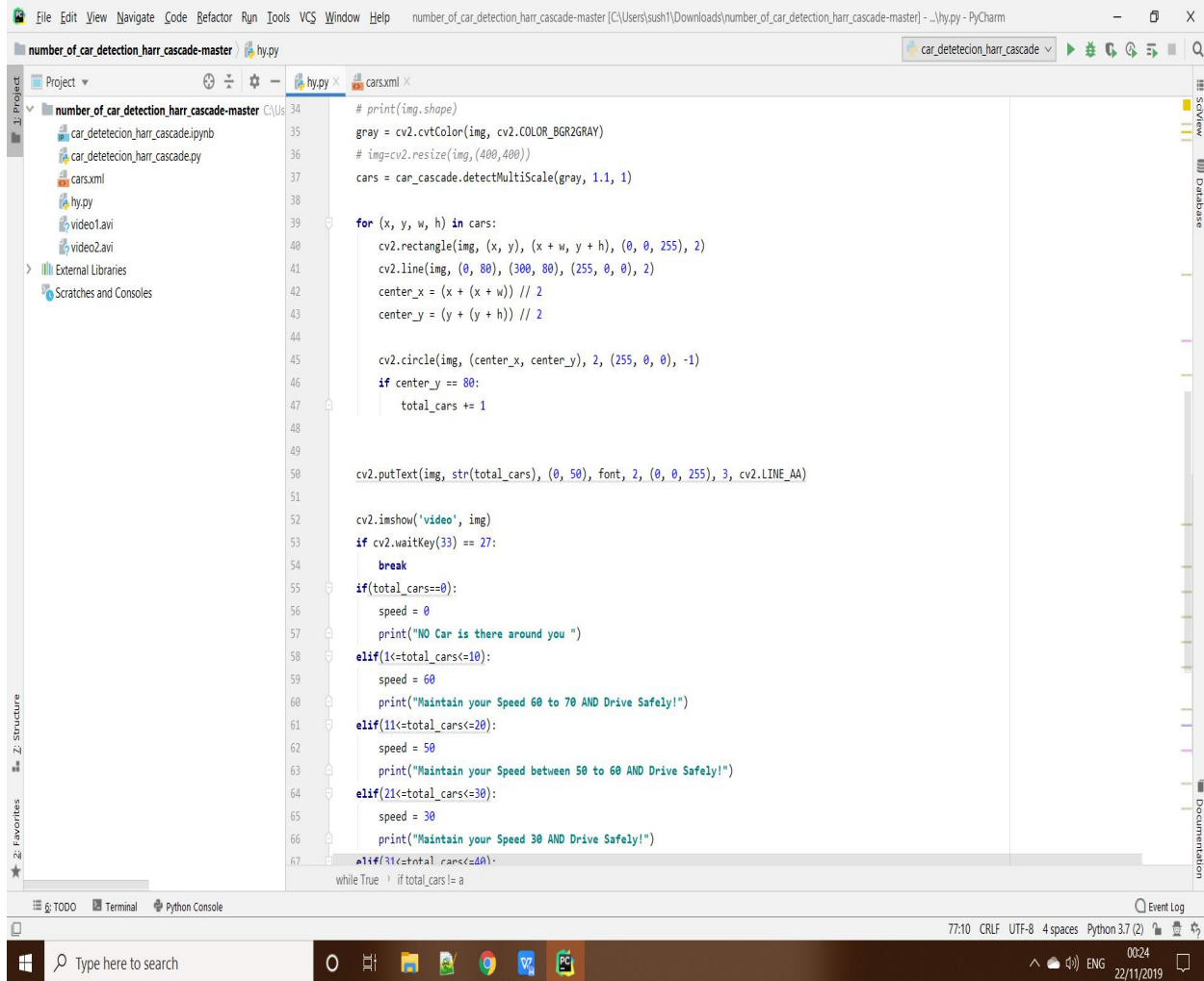
```
while 1:
```

```
    db.child('Attendance/M1').set(input('enter the value'))
```

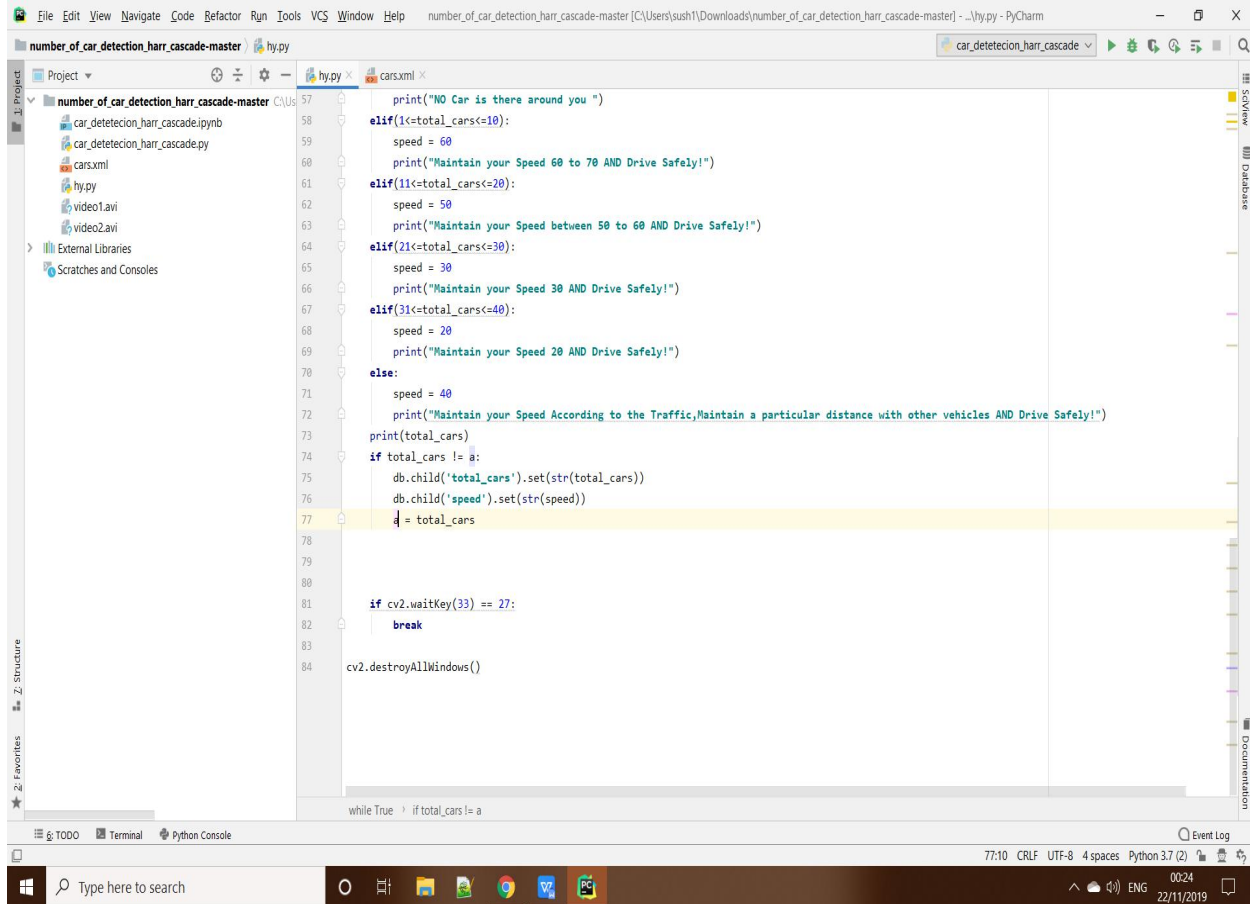
## SCREENSHOTS:



## Family Security

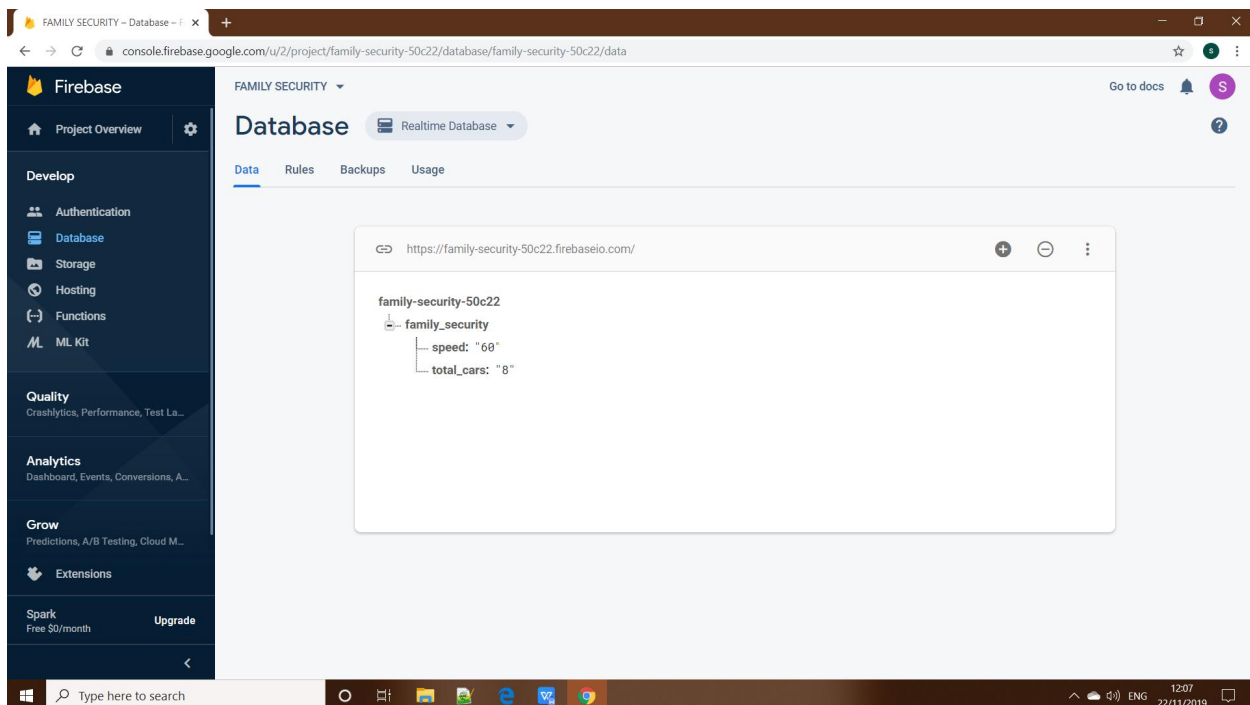


## Family Security



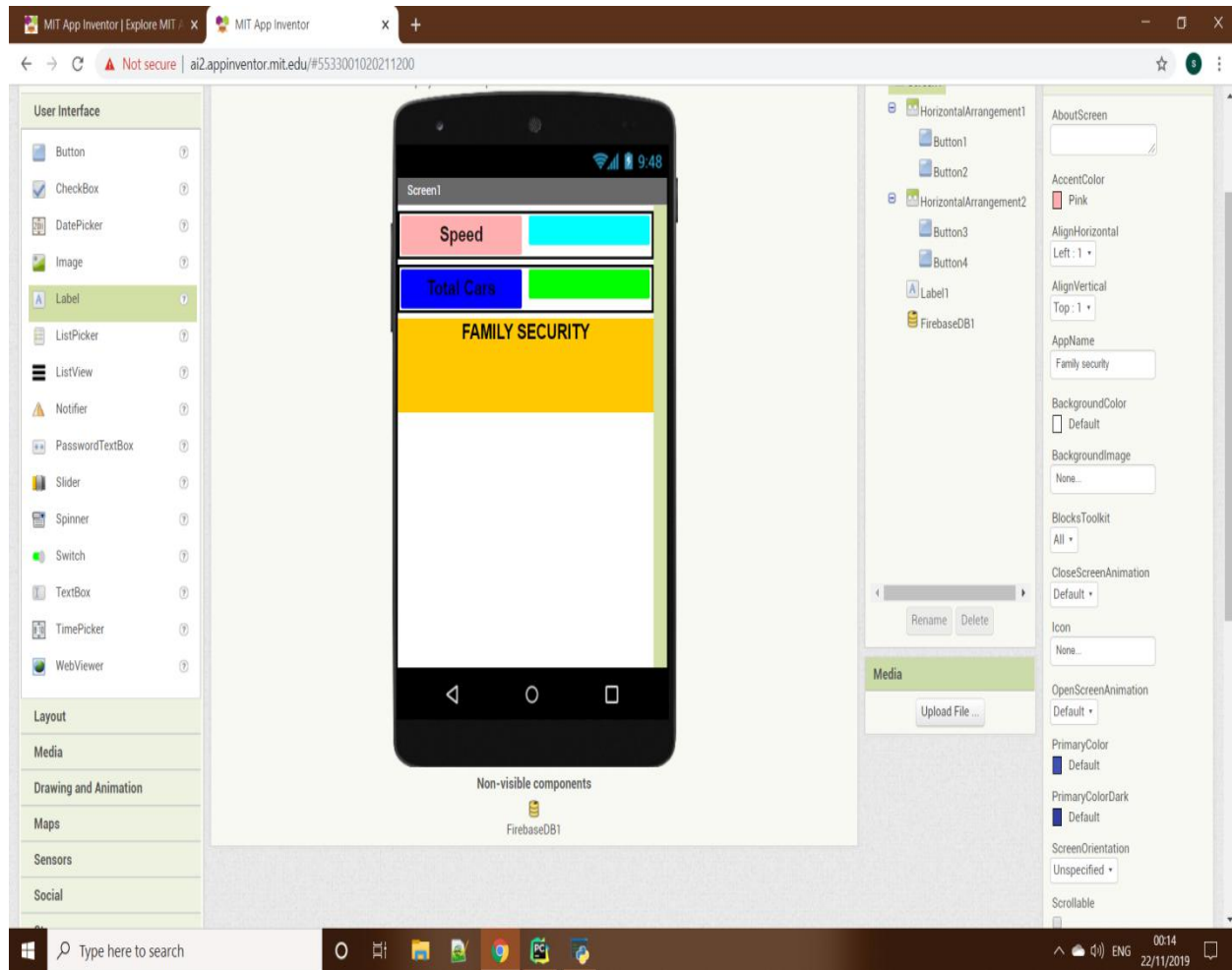
The image shows a PyCharm IDE window titled "number\_of\_car\_detection\_harr\_cascade-master". The main editor displays a Python script named "hy.py". The script uses OpenCV for video processing and a database to store car count and speed data. It features a series of conditional statements that print safety messages based on the number of cars and the current speed. The script also includes a loop that waits for a key press before destroying the video window.

```
57 print("NO Car is there around you ")
58 elif(1<=total_cars<=10):
59     speed = 60
60     print("Maintain your Speed 60 to 70 AND Drive Safely!")
61 elif(11<=total_cars<=20):
62     speed = 50
63     print("Maintain your Speed between 50 to 60 AND Drive Safely!")
64 elif(21<=total_cars<=30):
65     speed = 30
66     print("Maintain your Speed 30 AND Drive Safely!")
67 elif(31<=total_cars<=40):
68     speed = 20
69     print("Maintain your Speed 20 AND Drive Safely!")
70 else:
71     speed = 40
72     print("Maintain your Speed According to the Traffic,Maintain a particular distance with other vehicles AND Drive Safely!")
73 print(total_cars)
74 if total_cars != a:
75     db.child('total_cars').set(str(total_cars))
76     db.child('speed').set(str(speed))
77     a = total_cars
78
79
80
81 if cv2.waitKey(33) == 27:
82     break
83
84 cv2.destroyAllWindows()
```



## FIREBASE CONNECTED WITH APP AND ABOVE CODE

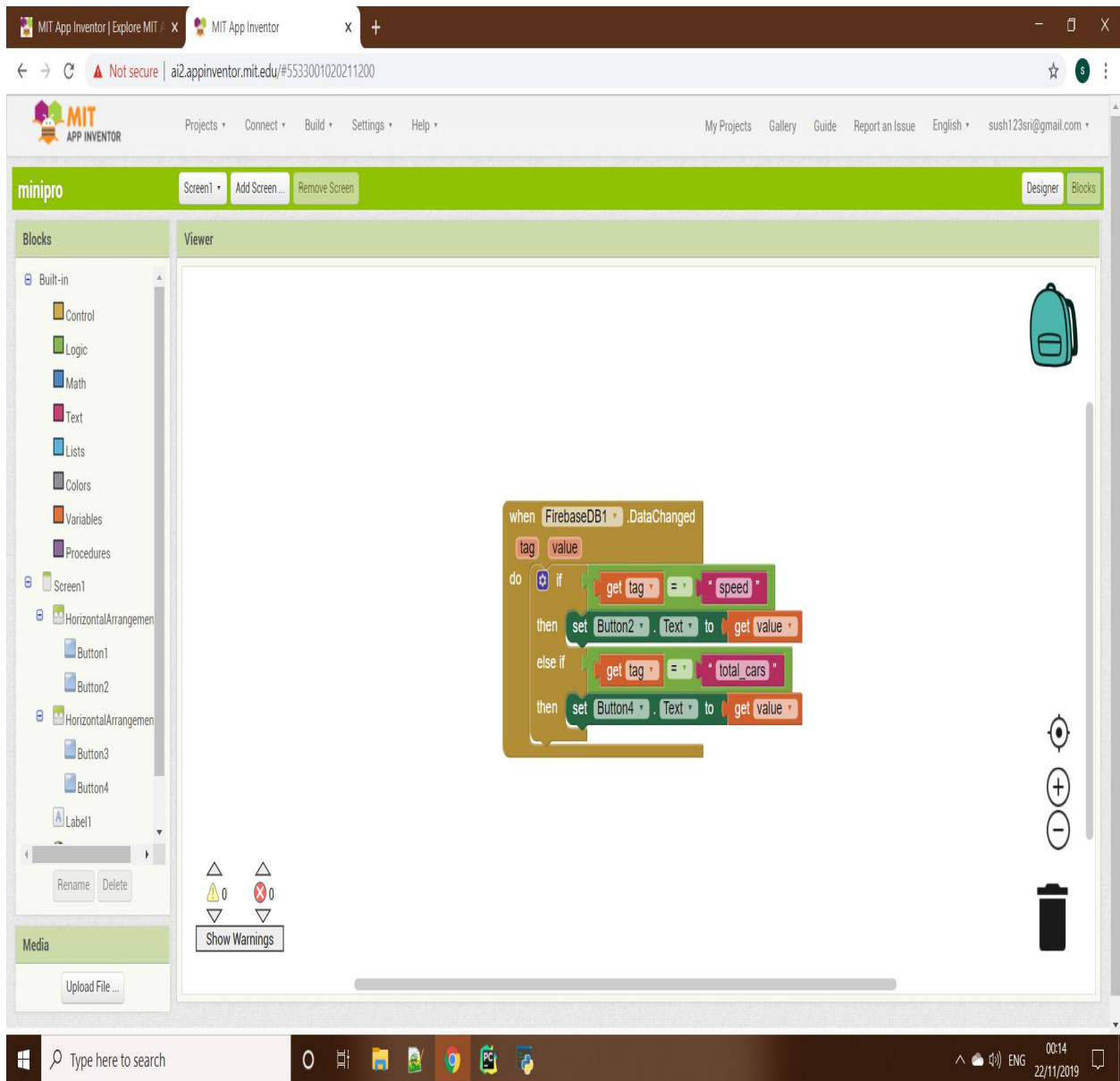
## Family Security



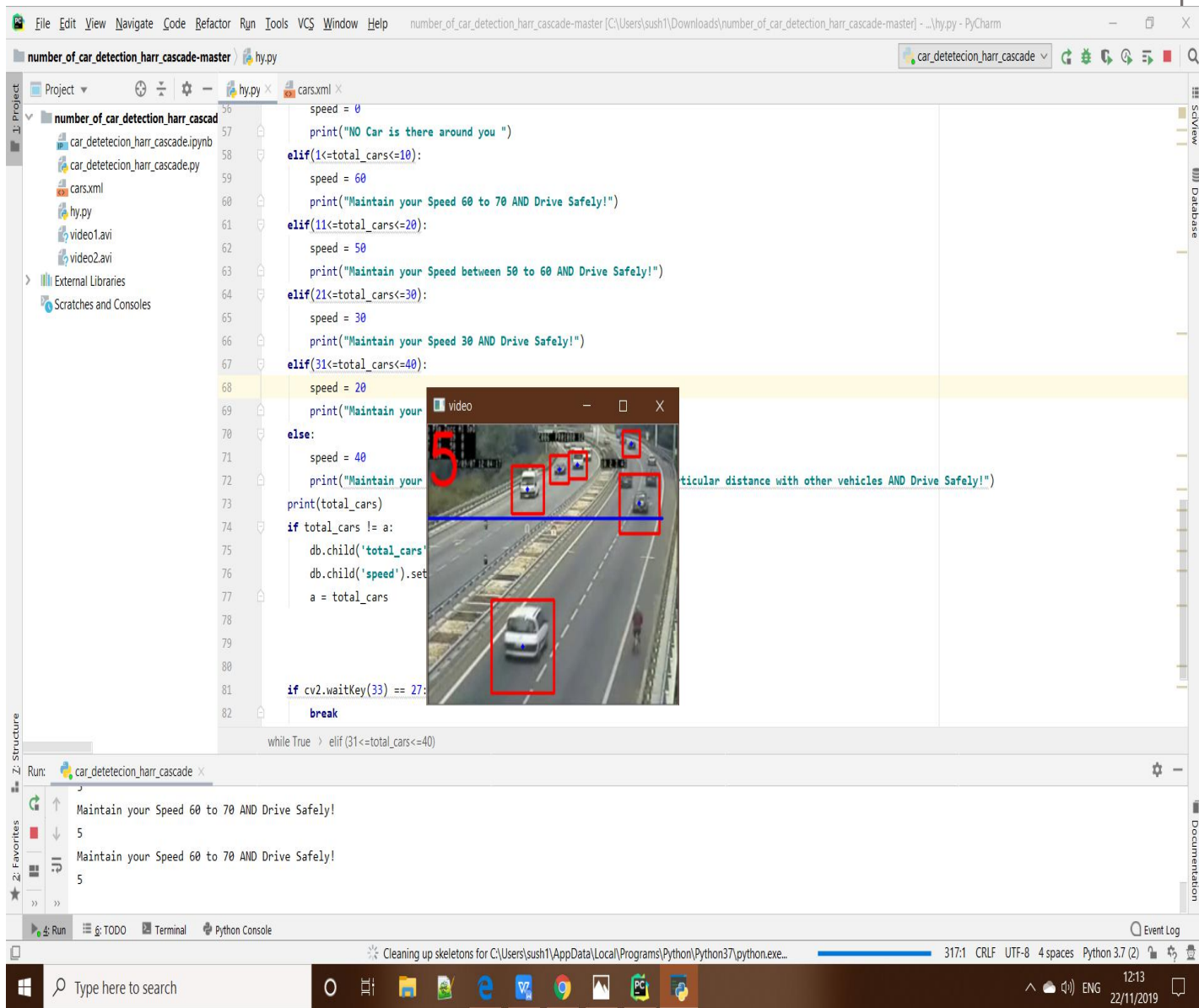
## APP IMPLEMENTATION USING APP INVENTOR



## Family Security



**PROGRAMMING DONE**

**OUTPUT:**

## **BIBLIOGRAPHY**

1. [www.google.com](http://www.google.com)
2. Geeksforgeeks
3. [www.edureka.com](http://www.edureka.com)
4. Mr. Amir khan sir