

# **VEHICLE DETECTION USING HAAR CASCADE**

**A PROJECT REPORT**

submitted by

**N SUBRAMANYA KUMAR ( 12BCE1019 )**

**K HARSHAVARDHAN ( 12BCE1113 )**

**M NARAYANA VIPUL ( 12BCE1114 )**

*in partial fulfillment for the award*

of

**B. Tech**

degree in

**Computer Science and Engineering**

**School of Computing Science and Engineering**



**April - 2016**

# **VEHICLE DETECTION USING HAAR CASCADE**

**A PROJECT REPORT**

submitted by

**N SUBRAMANYA KUMAR ( 12BCE1019 )**

**K HARSHAVARDHAN ( 12BCE1113 )**

**M NARAYANA VIPUL ( 12BCE1114 )**

*in partial fulfillment for the award*

of

**B. Tech**

degree in

**Computer Science and Engineering**

**School of Computing Science and Engineering**



**April - 2016**



## School of Computing Science and Engineering

### DECLARATION

We hereby declare that the project entitled “**Vehicle Detection Using Haar Cascade**” submitted by us to the School of Computing Science and Engineering, VIT University, Chennai Campus, Chennai 600127 in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** is a record of bonafide work carried out by us under the supervision of **Prof. Rukmani P.** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

**N SUBRAMANYA KUMAR ( 12BCE1019 )**

**K HARSHAVARDHAN ( 12BCE1113 )**

**M NARAYANA VIPUL ( 12BCE1114 )**



## School of Computing Science and Engineering

### CERTIFICATE

The project report entitled “**Vehicle Detection Using Haar Cascade**” is prepared and submitted by **N SUBRAMANYA KUMAR (12BCE1019), K HARSHA VARDHAN (12BCE1113), M NARAYANA VIPUL (12BCE1114)**. It has been found satisfactory in terms of scope, quality and presentation as partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering** in VIT University, Chennai Campus, Chennai, India.

**Prof. Rukmani P**  
**(Internal Guide)**

**Examined by:**

**Examiner 1**

**Examiner 2**

## ACKNOWLEDGEMENT

We would like to express our deepest appreciation and gratitude to all those who provided us the possibility to complete this report. We give our special gratitude to our faculty guide, **Prof. Rukmani P**, whose contribution in stimulating suggestions and encouragement, helped us to coordinate our project and writing this report.

Furthermore we would also like to acknowledge with much appreciation the crucial role of our **Program Chair Dr. Ganesan R**, whose encouraging words enabled us to complete this project successfully. We would also like to thank **Dean of the Department, SCSE, Dr. Jeganathan L**. whose constant word of encouragement helped us immensely during the course of this Project. Last but not least, we take this opportunity to thank **Prof. Bhargavi R**, whose timely mails, and proper communication, facilitated smooth operation and completion of project in stipulated time.

# CONTENTS

<b>Chapter</b>	<b>Title</b>	<b>Page</b>
	Title Page	i
	Declaration	ii
	Certificate	iii
	Acknowledgement	iv
	Table of contents	v
	List of Tables	vii
	List of Figures	viii
	List of Abbreviation	x
	Executive Summary	xi
1	Introduction	
	1.1 Objective	1
	1.2 Background	1
	1.3 Motivation	2
2	Project Description and Goals	3
3	Technical Specification	
	3.1 Hardware Specifications	5
	3.2 Software Specifications	5
	3.3 Data flow	6
4	Design, Approach and Details	
	4.1 Design Approach	8
	4.2 Codes and Standards	11
	4.3 Constraints, Alternatives and Tradeoffs	11
5	Schedule, Tasks and Milestones	14
6	Project Demonstration	19

7	Cost Analysis	25
8	Summary and Conclusion	26
9	References	27
	Appendix A – Schematic	29
	Appendix B – Requirements	32
	Appendix C – Source Code	33
	Appendix D – Instructions	50

## LIST OF TABLES

Table	Page
Table 4.1: Comparison of Alternatives of Raspberry Pi	11
Table 5.1 Project Schedule	13
Table 7.1: Component and Total Cost of the Project	22



# LIST OF FIGURES

<b>Title</b>	<b>Page</b>
Fig. 2.1: Block Diagram of the Project	4
Fig. 3.1: Data Flow of the Project	6
Fig. 4.1 Overall Engineering Design	7
Fig. 4.2 Pictorial Representation of Hardware	8
Fig. 4.3 Django MVT Architecture	9
Fig 5.1 Gantt chart Of the Project	17
Fig. 6.1 Functioning Embedded Device	20
Fig. 6.2 Bike detection	21
Fig. 6.3 Car detection	21
Fig. 6.4 Vehicle Counting	22
Fig. 6.5 Web application	23
Fig. 6.6 Website	24
Fig. 6.7 Email Subscription	24
Fig. 10.1 Raspberry Pi Schematic	26
Fig. 10.2 Raspberry Pi B board	27
Fig. 10.3 Raspberry Pi Schematic	27
Fig. 10.4 Logitech c260	28
Fig. 10.5 Samsung 32GB micro SDHC	28

Fig. 1.Creating a project	49
Fig. 2.Start server	49
Fig. 3.Create an app	49
Fig. 4.Migrating required databases	50
Fig. 5.Interactive shell console	50
Fig. 6.Creating a superuser	51

## LIST OF ABBREVIATIONS

Abbreviation	Expansion
Opencv	Open Source Computer Vision
SoC	System On Chip
IEEE	Institute of Electrical and Electronics Engineers
HDMI	High-Definition Multimedia Interface
MIMO	Multiple Input Multiple Output
SDHC	Secure Digital High Capacity
API	Application Programming Interface
IOT	Internet of Things
OS	Operating System
Wi-Fi	Wireless Fidelity
MVC	Model-View-Controller
MVT	Model-View-Template
SDK	Software Development Kit
UI	User Interface
MP	Mega Pixel
USB	Universal Serial Bus
LED	Light Emitting Diode
GB	Gigabyte

## **Executive Summary**

The number of vehicles is increasing faster than the number of new roads being built, resulting in experiencing heavy traffic. Due to the current traffic management and increasing number of vehicles, a lot of man hours are being wasted on the roads. This project helps effectively in communicating with the people about the traffic details. In this project we have designed and developed a device capable of detecting on road vehicles and classify them. This was achieved by using the data collected in the past for training the cascade classifier. All the data collected from the raspberry pi is stored on to a database. We have developed a web application as the user interface to access the collected data. This includes developing a map module using Mapbox. This project collects the live feed from the camera and is analyzed for vehicles. When a vehicle is moving a rectangular box is shown around the vehicle indicating it has been detected. Different colors are used for rectangular box to classify the types of vehicles. A count to the vehicles is made available. The incorporated application shows the current traffic conditions of the particular area. The detection and classification module with xml cascades trained was one of major contribution to project. Other significant contributions include the development of web application which can provide the data collected by the prototype device on to map module with greater visualization in real time. The web application can be improved by analyzing the past traffic data and provide graphical data to predict the future patterns. As technology updates, new cars arrive on roads, so it is recommended to update training data constantly. Training the data is recommended for better accuracy of the experiment. The future work includes implementation of the project at multiple places for better knowledge about the traffic conditions. The web application can be improved by analyzing the past traffic data and provide graphical data to predict the future patterns.

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 Objective**

The primary objective for the project is to design and develop a device capable of detecting on road vehicles and classify them based on type of vehicle using the camera. Collection of the past camera data and use it for training of vehicle detection. To develop a web application for the user interface that provides the details about the traffic conditions.

### **1.2 Background**

Visual tracking is an important task within the field of computer vision. The proliferation of high-end computers, the availability of high quality video cameras, and the increasing need for automated video analysis have generated a great deal of interest in visual tracking algorithms. The state of this art has advanced significantly in the past 30 years. Generally speaking, the use of visual tracking is pertinent in the tasks of motion-based recognition, automated surveillance, video indexing, human–computer interaction and vehicle navigation, etc.

Visual tracking is a fundamental task in many computer vision applications and has been well studied in the last decades. Although numerous approaches have been proposed, robust visual tracking remains a huge challenge. Difficulties in visual tracking can arise due to abrupt object motion, appearance pattern change, non-rigid object structures, occlusion and camera motion.

Object tracking is a decades old computer vision problem with a variety of subtasks to be addressed. It is naturally suited to use both geometry and appearance information. Recently, however, advances in computing power have allowed much more sophisticated algorithms to track at or near frame rates. One can categorize this variety of approaches in several ways, although the latest trackers tend to combine several concepts or techniques. All must choose an appearance model to represent the positive and

negative images – the type of features used to describe the object, and distinguish it from negative images. This algorithm is known as Haar Cascade.

Historically, working with only image intensities made the task of feature calculation computationally expensive. A publication by Papageorgio discussed working with an alternate feature set based on Haar wavelets instead of the usual image intensities. Viola and Jones adapted the idea of using Haar wavelets and developed the so-called Haar-like features. A Haar-like feature considers adjacent 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.

### **1.3 Motivation**

In recent years, traffic congestion has become a significant problem. The number of vehicles is increasing faster than the number of new roads being built, resulting in experiencing heavy traffic. Due to the current traffic management and increasing number of vehicles, a lot of man hours are being wasted on the roads. Early solutions attempted to lay more pavements to avoid congestion, but adding more lanes is becoming less and less feasible. The quest for better traffic information, and thus, an increasing reliance on traffic surveillance, has resulted in a need for better vehicle detection. This project helps effectively in communicating with the people about the traffic details.

## **CHAPTER 2**

### **PROJECT DESCRIPTION AND GOALS**

#### **2.1 Description**

This project proposes a prototype of vehicle detection system, with various personalized features. These features include –camera captures the vehicles passing through the given area, classifying based on the type of vehicle, and counting them accordingly. The features added to this application are done by analysis of the data collected in the past to train the sample data. This information is sent to the database and is made available to the user in form of a web application. This web application serves as the user Interface. It also displays the traffic intensity at that time at a particular place. The detection is done using Haar Cascade algorithm. The software coding uses the Opencv image processing application in python.

Map module is incorporated in the web application using Mapbox which displays count and type of vehicle. We have adapted Django framework to build the web application. The web application provides features like subscription for newsletter, visualization on map and other details about the traffic. The user can register by providing their e-mail address on our website. All the registered users will receive the updates about the traffic conditions from the location. The email contains the number of cars and bikes. The email consists of details about traffic at the place.

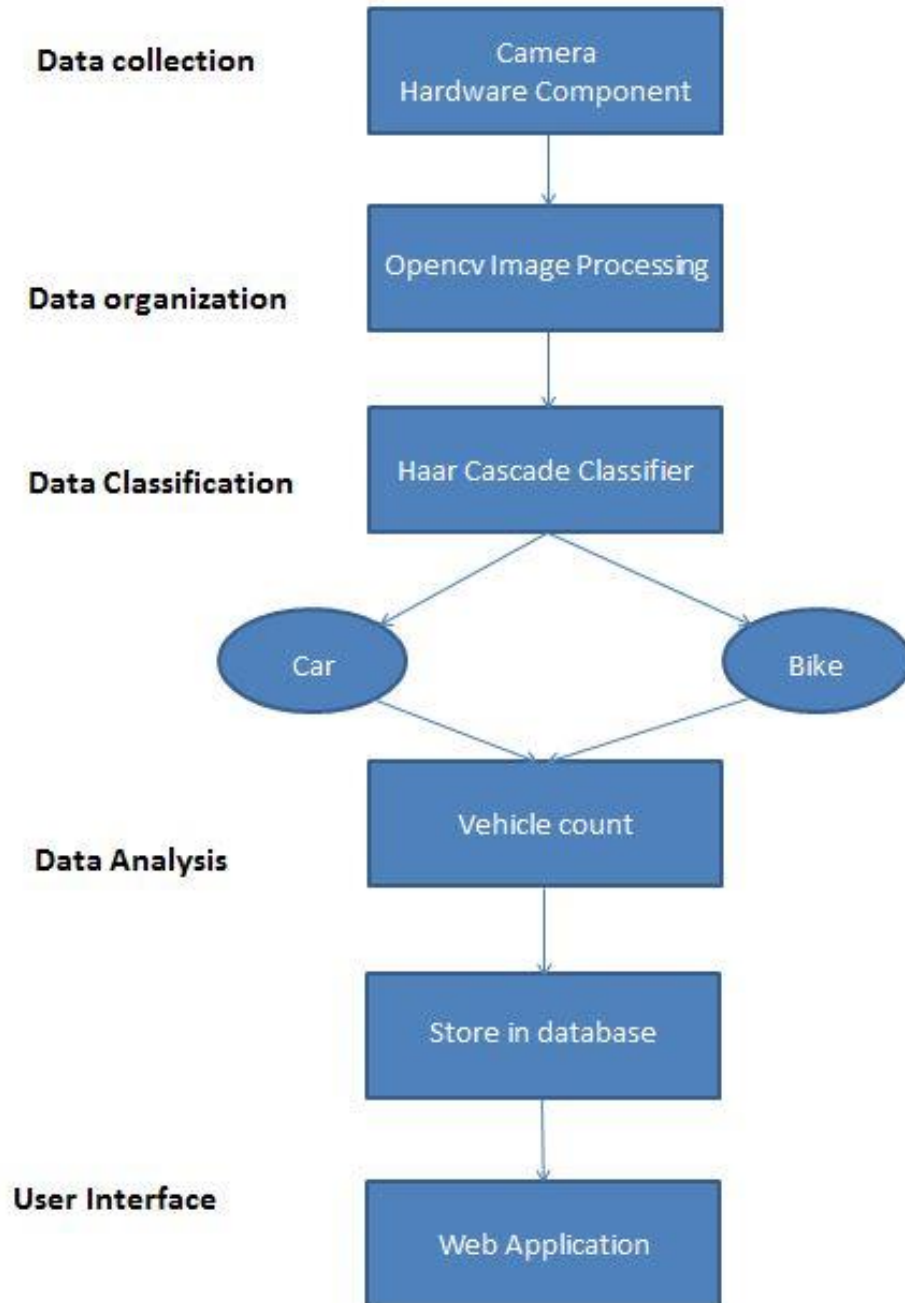


Fig. 2.1: Block Diagram of the Project

## 2.2 Goals

The project will result in the development of a prototype device with the capability of detecting and classifying vehicles. It will send the details to the web application. These readings are stored in the database and can be used to detect the traffic conditions. The stored details are displayed on map for users.



## **CHAPTER 3**

### **TECHNICAL SPECIFICATION**

#### **3.1 Hardware Specification**

##### **3.1.1 Raspberry Pi 2**

1. Model: B
2. SoC: Broadcom BCM2836
3. Memory: 1GB
4. Dimensions: 85.60 mm × 56.5 mm (about 3.2 x 2.1-inch)
5. Weight: 45g (1.6oz)
6. Power: 800 mA (4.0W)
7. Ports: 4x USB 2.0, 100BaseT Ethernet, HDMI, Micro SD card

##### **3.1.2 Camera**

1. Company: Logitech
2. Model: C260
3. High-definition: 720p
4. Photo Quality: 3MP

##### **3.1.3 SD Card**

1. Class: EVO Grade 1, Class 10
2. Type: Micro SDHC
3. Speed: 48MB/s
4. Memory: 32GB
5. Brand: Samsung

### 3.2 Software Specifications

1. Django Framework
2. Mapbox API
3. Windows IoT core OS
4. SQLite database
5. DVDVideosoft
6. Virtualdub
7. Python 3.5.1 with
  - a. Opencv - Opencv is a real time computer vision library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, and moving objects.

### 3.3 Data Flow Diagram

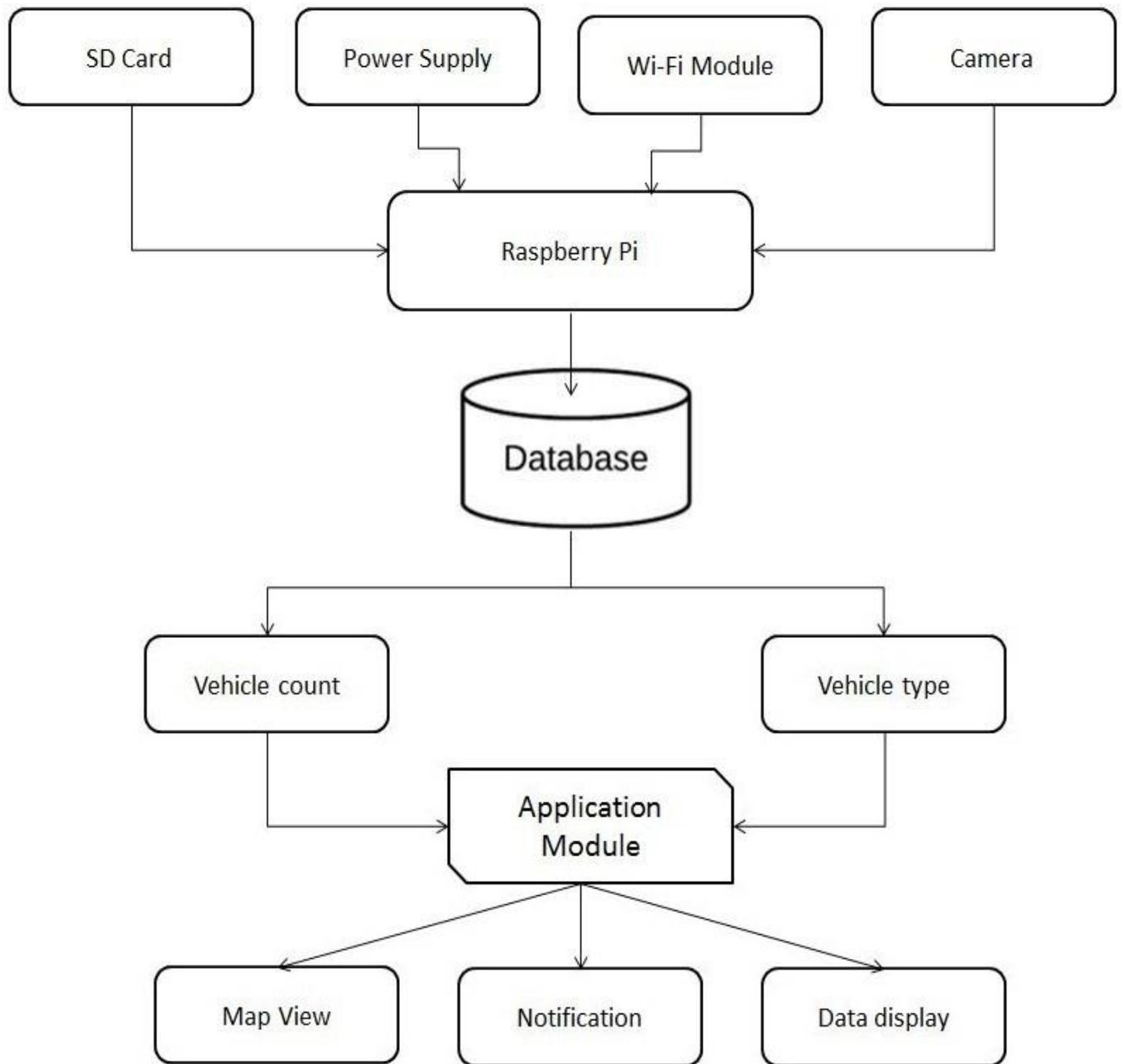


Fig. 3.1: Data Flow of the Project

## CHAPTER 4

### DESIGN, APPROACH AND DETAILS

#### 4.1 Design Approach

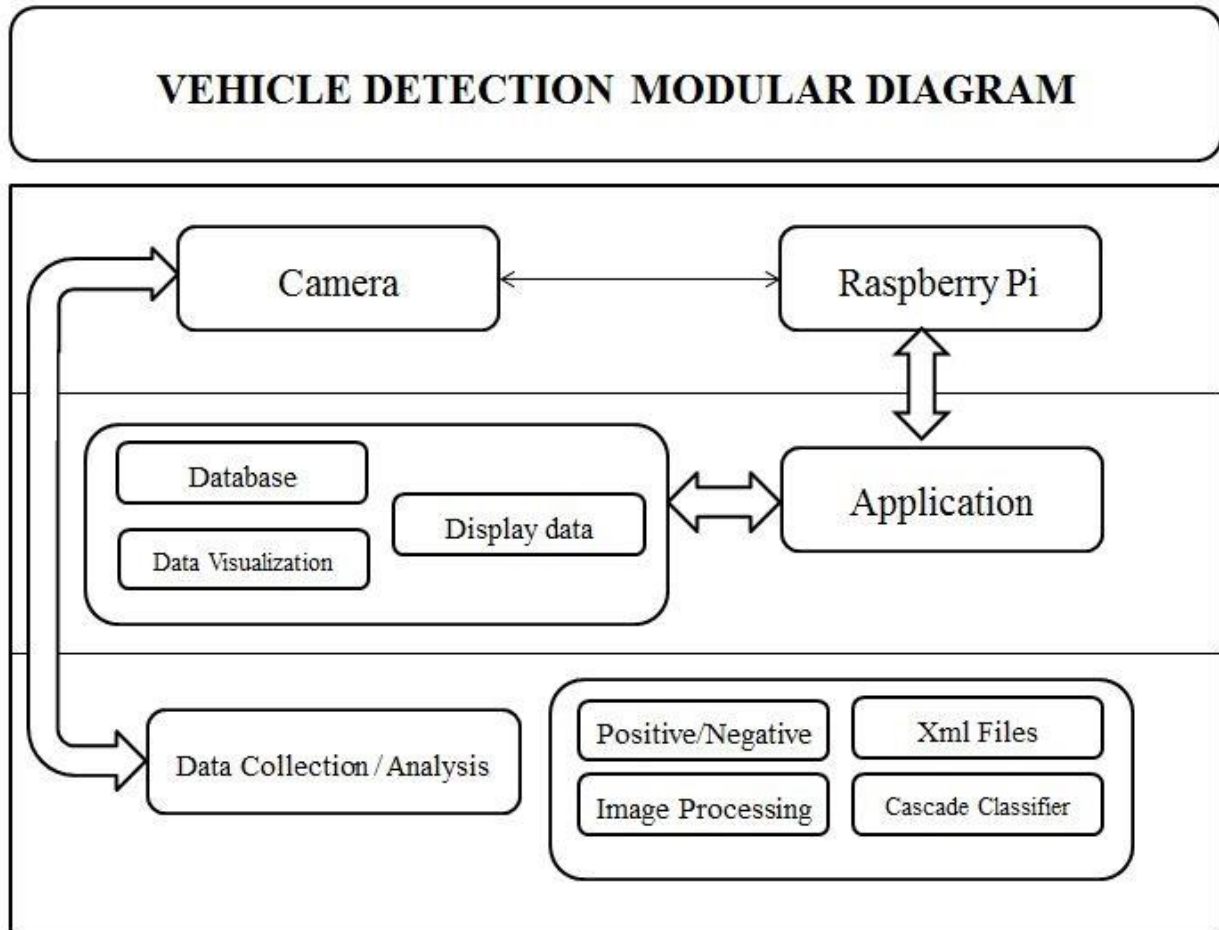


Fig. 4.1 Overall Engineering Design

##### 4.1.1 Hardware Design

The hardware module incorporates raspberry pi with camera which collects the data to be analyzed. SD card connected to raspberry pi is used for data storage. Wi-Fi module provides internet connection. The analyzed data is sent to the server which can be used for web application.

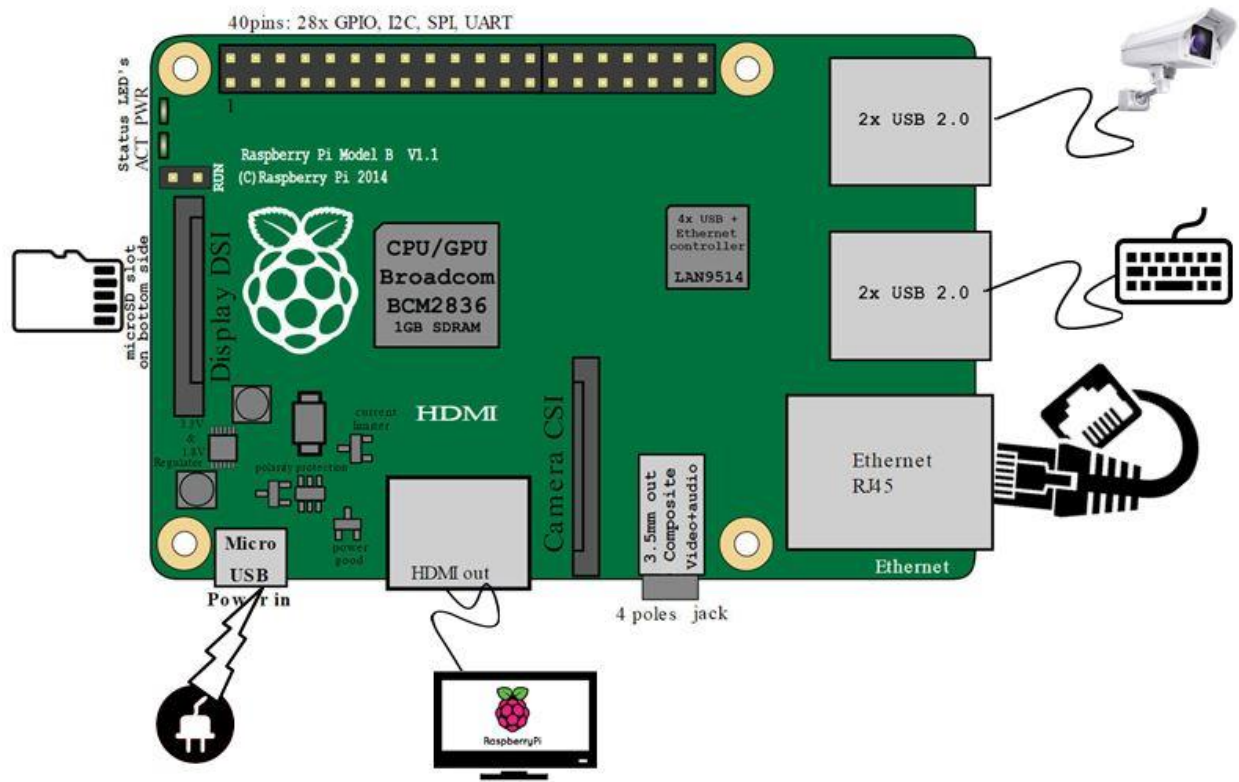


Fig. 4.2 Pictorial Representation of Hardware

#### 4.1.2 Software Design

First we have to collect the past traffic camera footage for the sample data to be used for the vehicle detection. By default, the data collected is in the form of video, which has to be converted to images. Dvdvideosoft can be used for this conversion. This conversion can be done based on the frame division or time division. Accuracy will be more if the division is done based on the frame rate.

The images are then divided into positive and negative images manually or by brute force method. A positive image contains the vehicle which we want to recognize. Negative images do not contain the vehicle in it. All background images without any vehicle come under negative images. Any images with vehicle should be considered as positive image only. The accuracy can be increased by collecting more number of positive and negative images. All the positive and negative images should be stored separately. A text file with all the names of negative images is

created. Absolute path is added to all the text file names. Create a vector file for the positive images using ‘opencv\_createsample’ utility. Using ‘opencv\_traincascade’ utility classifier is trained, generating the xml files about the types of vehicles. All the generated xml files are imported into python and using Opencv - counting and classification is done. All the data is then sent to the database using the Wi-Fi module attached to raspberry pi. After training is done, the project is deployed to the raspberry pi for independent use.

#### 4.1.3 Web Application

The web application provides the user with an interface to know the details of roads. This application is made using Django framework. Django uses MVC architecture pattern. The Model is responsible for interaction with the database. The application provides the real time data about traffic including type of vehicles and number of vehicles passing thereby informing the user about current traffic intensity.

The application provides interface where the traffic details are visualized on a map. It also shows a notification with the intensity of traffic. All the details are stored and retrieved from the database. Once the vehicles are detected, an entry to the database is done. Based on the entry into the database, vehicles are displayed on the map. The map coordinates for the given area is already stored in the database for easy update of details.

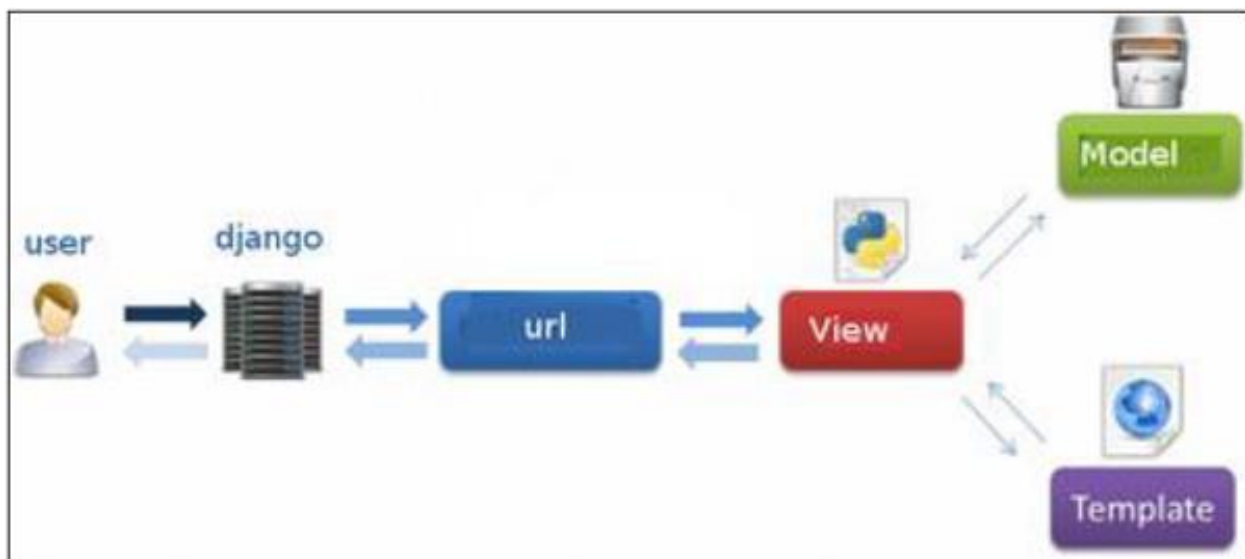


Fig. 4.3 Django MVT Architecture

## 4.2 Codes and Standards

1. IEEE Standard for IoT
2. Image processing Standards (RGB)
3. IEEE 802.11 b/g/n

802.11n was designed to improve on 802.11g in the amount of bandwidth supported by utilizing multiple wireless signals and antennas (called MIMO technology) instead of one. Industry standards groups ratified 802.11n in 2009 with specifications providing for up to 300 Mbps of network bandwidth. 802.11n also offers somewhat better range over earlier Wi-Fi standards due to its increased signal intensity, and it is backward-compatible with 802.11b/g gear.

This standard is employed by the Raspberry pi which is responsible for transmitting data from the device to the server using the Wi-Fi module attached to it.

## 4.3 Constraints, Alternatives and Tradeoffs

### 4.3.1 Realistic Constraints

1. Economic constraints: Our project requires conversion of video to image and it requires high multimedia processing power for video analysis, it costs far more than a basic Arduino board. Depending upon the type of quality required, camera costs are varied. More the quality, more the costs for camera. Average costs for 1080p camera is Rs.4000.
2. Ethical Constraints: This project helps in reducing difficulties for travellers, and save lot of time on roads. So, this method can be implemented anywhere using the Pseudo code and design documents that are available as open source. In this way it helps anyone to download and use it.
3. Manufacturability: This device can be manufactured by the assembly of easily available components and we can deploy the source code to the raspberry pi. The collection of trained data is difficult, as sample data will be different for places.

4. Sustainability: As the project deploys real time traffic data on web application, Internet is required for transfer of data between device and server. The device and camera should be connected to power supply for continuous working.
5. Usability: The project provides a real time application interface that provides key insights to user about traffic environment. However many places are not added to maps, users choice becomes redundant. The application is just like visting any other website which doesn't need any learning. Google map integration to the application makes it much more user friendly.

#### 4.3.2 Design Constraints

1. Due to cost constraints, the camera used is not very expensive and therefore cannot record high definition images and better accuracy.
2. Connection Reliability: Real time traffic data cannot be shown on the application without internet, so internet connection is must.
3. As we have used .bmp format for image training, all the new images should be converted to the same format before training the classifier.

#### 4.3.3 Component Alternatives

##### Raspberry Pi Alternatives

Component	Advantages	Disadvantages
Arduino Uno	Beginners Board with Replaceable microprocessor. Cost is lower than Raspberry Pi.	Less I/O ports as compared to Leonardo. Programmable with USB.
Beagle Bones	Faster Processor with large memory. Uses Linux distribution. 66 Digital I/O along with 7 Analog I/O ports	Much higher cost as compared to Arduino or raspberry pi.
Intel Galileo	Compatible with other Arduino boards and LED for status indication.	No video support and multimedia processor

Table 4.1: Comparison of Alternatives of Leonardo



Amongst the alternatives Beagle Bones would be the ideal replacement of Raspberry Pi. However despite of presence of a very large array of I/O ports almost all of them are digital I/O ports rather than analog port. The cost for Beagle Bones is much higher than Raspberry pi. Apart from this, the processor speed along with extra memory helps in analyzing the video easily and transmits the recorded data to the server.

#### **4.3.4 Tradeoffs**

All the vehicles are analyzed and detected, and then they are added to the database for vehicle count. Even the accuracy is good; some vehicles may not be detected due to their absence in positive images. To overcome this, all the new vehicles images should be added for classification. Constant update to the positive images should be done.

Although the accuracy is based on the training, number of positive and negative images should be higher for better training of sample. This results in best accuracy of the detection of vehicles.

## CHAPTER 5

### SCHEDULE, TASKS AND MILESTONES

#### 5.1 Task Schedule

Tasks	Starting Date	Completion Date	Assignee	Description	Number of hours for the task(per day)
Project Brainstorming	18 November 2015	01 December 2015	Entire Team	No proper insight and papers were available	8
Literature survey Partial	02 December 2015	12 December 2015	Entire Team	Searching for different methods for project.	4
Literature Survey Complete	13 December 2015	10 January 2016	Entire Team	Identified the different implementation for the project, along with benefits and drawbacks of each approach.	6
Module Separation and Assignment	11 January 2016	11 January 2016	Entire Team	Module Separation based on the members interest.	5

Assessment of components based on the cost.	12 January 2016	22 January 2016	Subramanya Kumar	Identifying different components for the project	5
Final Selection of Component based on cost and requirements of the project.	23 January 2016	27 January 2016	Subramanya Kumar	Finalizing the components and ordering them.	5
Ordering Component	28 January 2016	20 February 2016	Subramanya Kumar	Components ordered(camera Delayed in shipping)	4
Application Platform Identification	12 January 2016	14 January 2016	Narayana Vipul	Determining where the project must be implemented	5
Setting up SDK	15 January 2016	20 January 2016	Narayana Vipul	Setting up development environment for project	4
UI component design for the Project	21 January 2016	31 January 2016	Narayana Vipul	Started discussing design of the application	6
Assessment of Availability methods for visualization platforms and	12 January 2016	18 January 2016	Harsha Vardhan	Reading documentations and available integration possible	6

and analysis Platform					
Understanding Different methods and assumptions and effects on data.	19 January 2016	26 January 2016	Harsha Vardhan	working out maths on paper	6
Final Selection of algorithm	27 January 2016	29 January 2016	Harsha Vardhan	Finalizing the algorithm based on requirements of the project	4
Study configurations of h/w & s/w	28 January 2016	22 February 2016	Entire Team	Studied papers about different configurations	5
Report Submission for review 1	23 February 2016	26 February 2016	Entire Team		5
Collection of Footage	04 March 2016	08 March 2016	Entire Team	Permission from vice chancellor	4
Get Images from Video	09 March 2016	12 March 2016	Entire Team	Converting video to frames	5
Deciding on preprocessing techniques and acquiring a reliable test Data	13 March 2016	29 March 2016	Harsha Vardhan	Learning about image pre-processing techniques	6

Deciding on libraries and reading about their various functionalities	13 March 2016	17 March	Subramanya Kumar	Data type compatibility between libraries and data available	4
Organizing Images	18 March 2016	25 March 2016	Subramanya Kumar	Creating positive and negative images	6
Coding on with Different preprocessing and methods and testing	30 March 2016	03 April 2016	Harsha Vardhan	Code for analysis and opencv detection libraries	6
UI implementation of application module	13 March 2016	03 April 2016	Narayana Vipul	Developing basic app module	5
Report Submission for review 2	04 April 2016	08 April 2016	Entire Team		4
Implementing Django	09 April 2016	13 April 2016	Narayana Vipul		5
Data base integration with app module	13 April 2016	14 April 2016	Narayana Vipul	Database for data storage	4
Adding maps Module	25 March 2016	07 April 2016	Subramanya Kumar	Adding Google maps to app	5
Gathering data	07 April	13 April	Subramanya	Creating xml	4

	2016	2016	Kumar	files for training	
Training Classifier	13 April 2016	14 April 2016	Harsha Vardhan	Training Haar Cascade classifier	6
Making Modification in application to Incorporate added features	15 April 2016	20 April 2016	Harsha Vardhan	Moved to mapbox module	5
Re-coding according to new changes	15 April 2016	20 April 2016	Narayana Vipul	Changed code for new features	6
Running tests and identifying bugs	21 April 2016	22 April 2016	Entire Team		4
Integration with Raspberry Pi	23 April 2016	24 April 2016	Entire Team	Hardware integration	5
Draft Report Submission	25 April 2016	25 April 2016	Entire Team		6

Table 5.1 Project Schedule



Fig 5.1 Gantt chart Of the Project

## CHAPTER 6

### PROJECT DEMONSTRATION

#### 6.1 Hardware Device

The hardware comprises of Raspberry pi, camera, SD card and Wi-Fi module. The camera is primarily responsible for vehicle detection. The SD card stores all the xml trained classifier data. The Wi-Fi module transmits the analyzed data to the web interface. This interface aids in identifying the traffic conditions and visualize them on map module in real time.



Fig. 6.1 Functioning Embedded Device

#### 6.2 Vehicle Detection

The picture represents the running application for detection of vehicles. When the vehicle is passed through the given area as recorded by the camera, it shows a rectangular box as an indication for the positive detection of vehicle. When a bike is detected the color of the rectangular box is shown as blue, whereas for car it is detected by a green



colored box. The color for this rectangular box can be selected based on the user preference. The rectangular box position in the image indicates where, exactly a match with the trained classifier is found.



Fig. 6.2 Bike detection

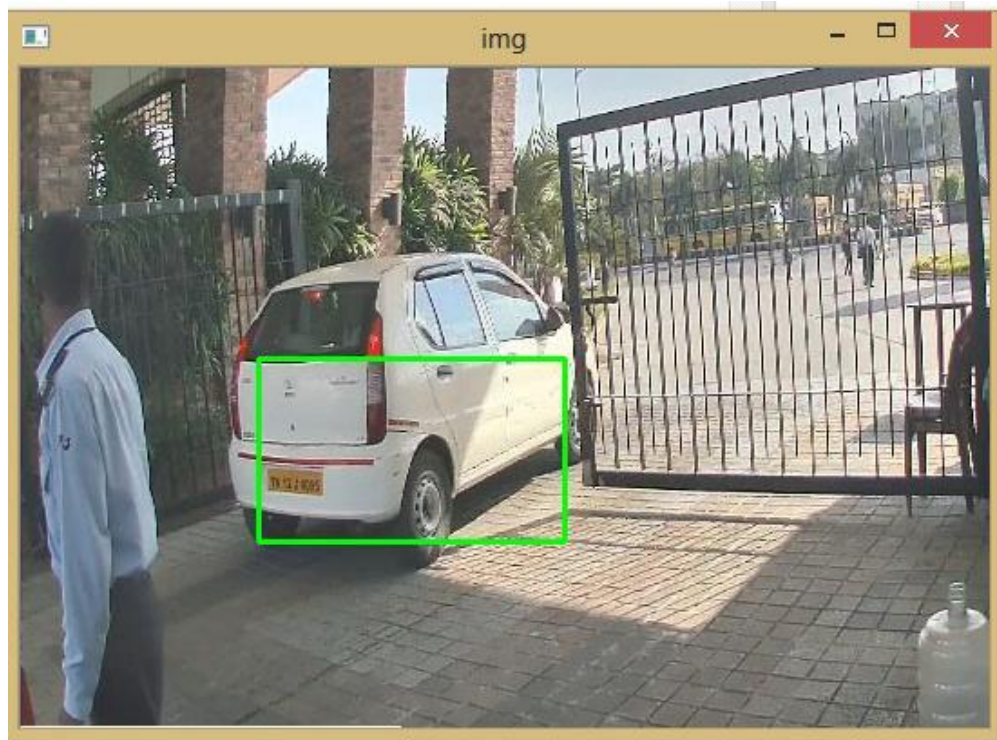


Fig. 6.3 Car detection

### 6.3 Vehicle Counting

The picture represents the running application for counting vehicles and their classification. As the vehicle crosses the line present at the center of the video, the vehicle is counted and stored in the database. The vehicle detected is classified as car based on the trained xml files. The count cannot be added to the database if it doesn't cross the line even after the vehicle is detected.

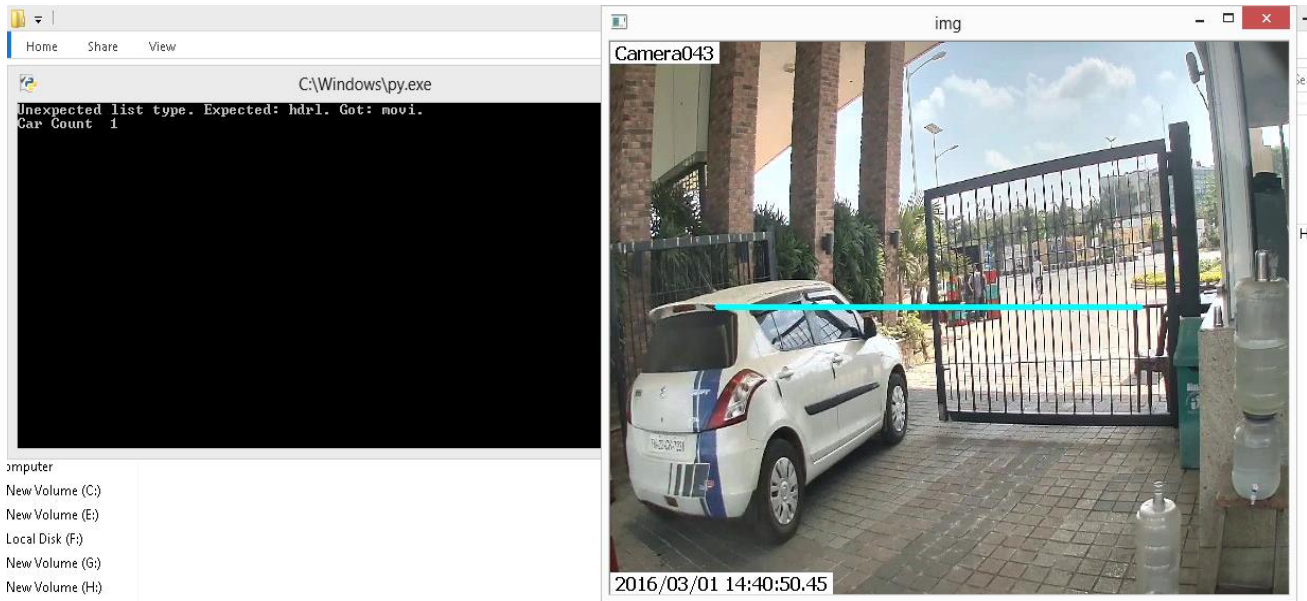


Fig. 6.4 Vehicle Counting

### 6.4 Application Module

The picture representing the application module, where the data collected is accessed into web interface is shown below. The application retrieves the data from the database and for visualization. The map module is done using the Mapbox api. When a car is passing through the place, a popup indication with a car symbol is shown. The same is shown for bikes with a different symbol on it. Whenever a vehicle crosses that gate, the map starts showing the vehicle movement for certain distance. A legend part is

used in the map to display the count for vehicles. A separate count is shown for bikes and cars. The movement shown on map is only for visualization, it is not based on vehicle movement. Different types of color and shapes are used to differentiate vehicles based on classification.

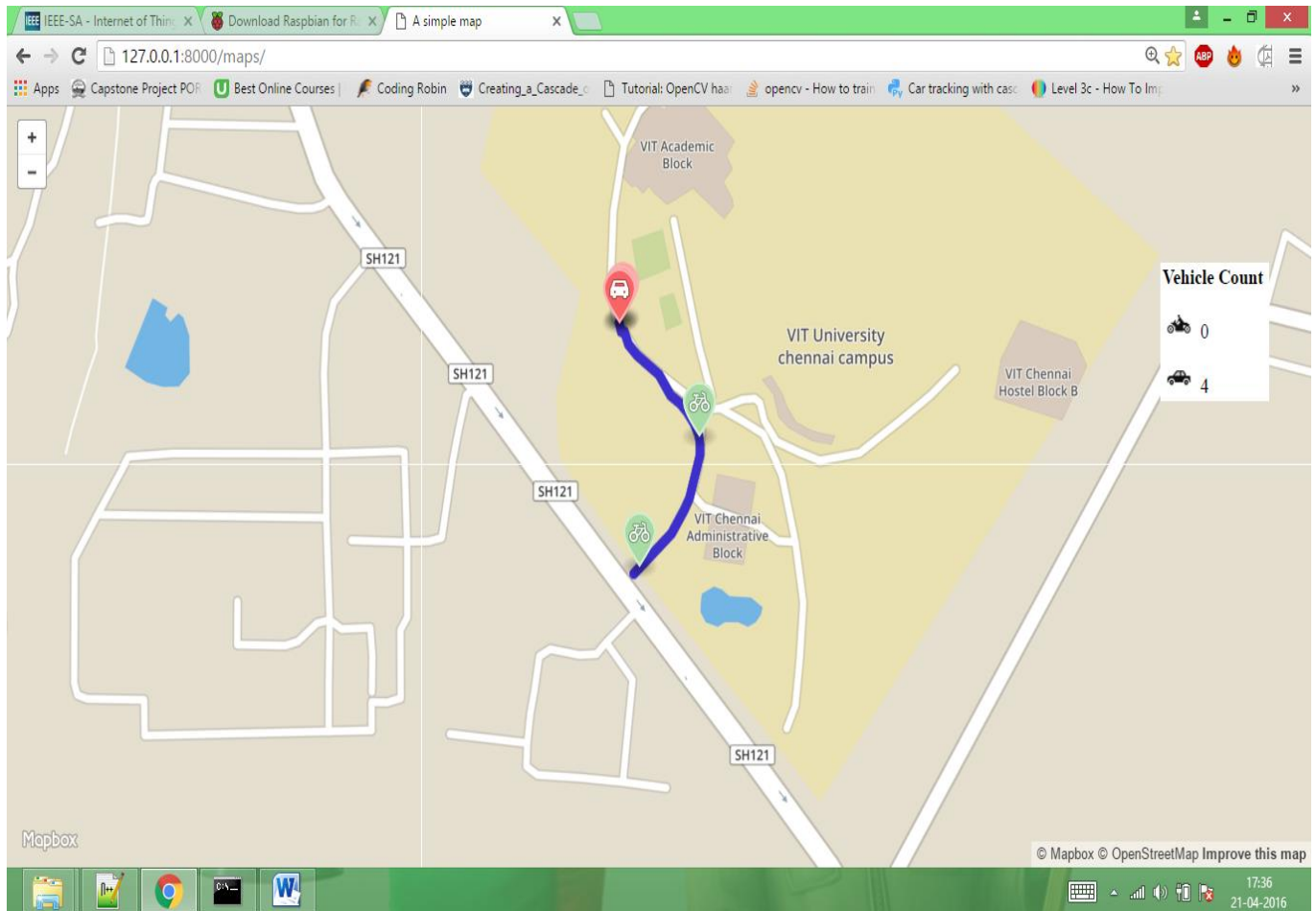


Fig. 6.5 Web application

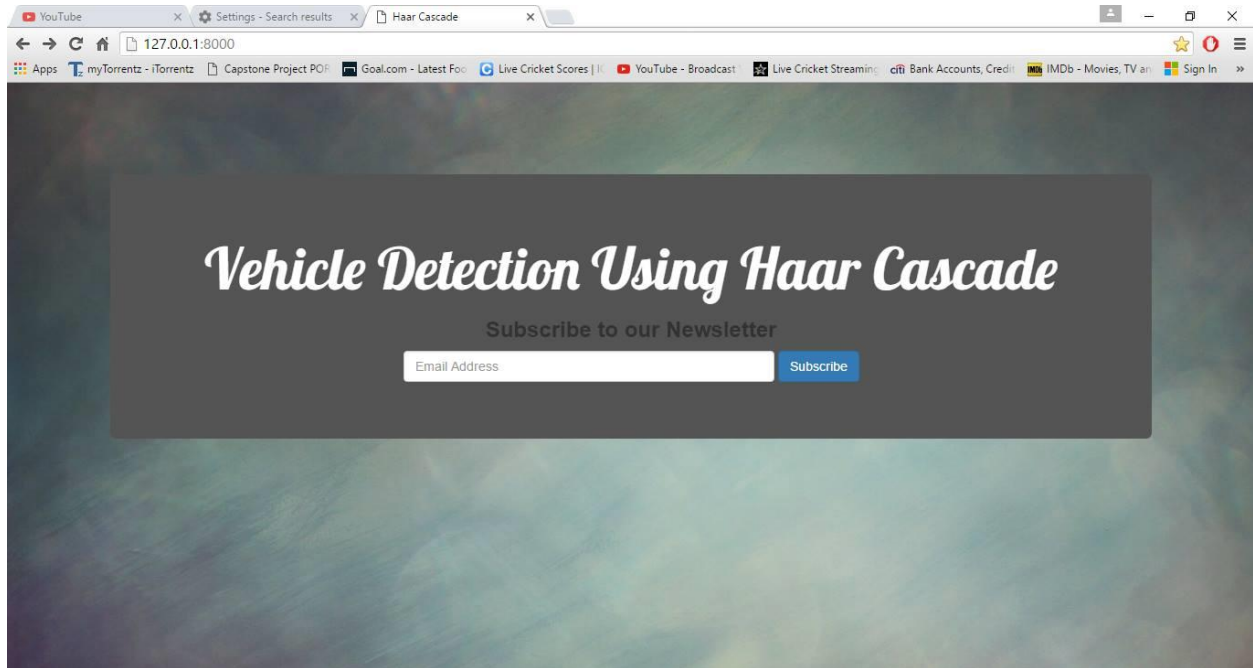


Fig. 6.6 Website

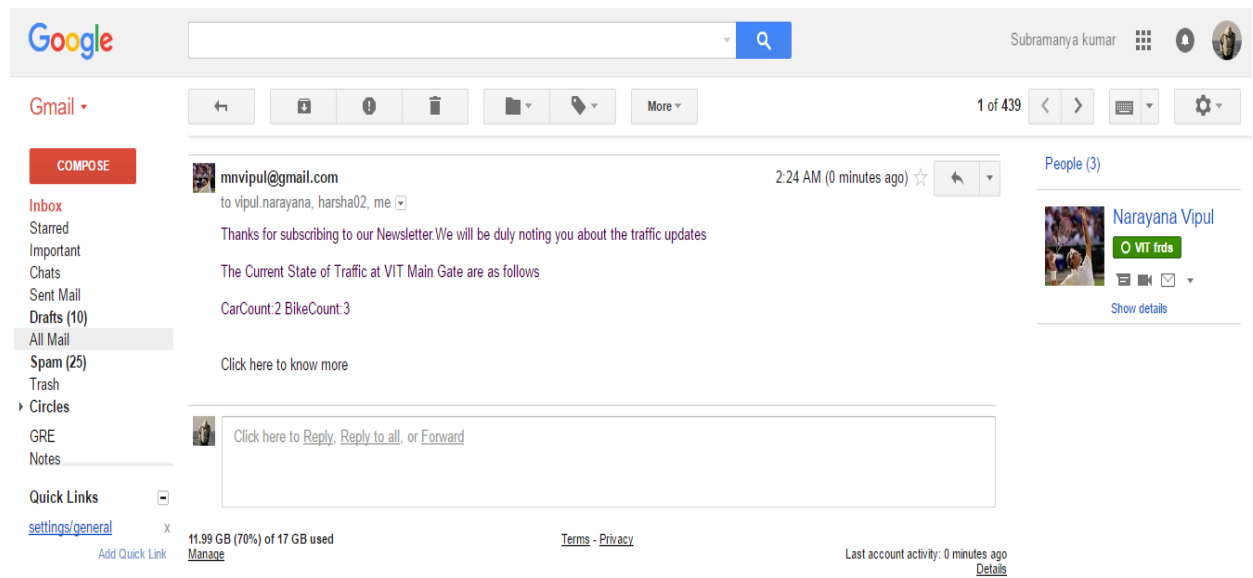


Fig. 6.7 Email Subscription

## CHAPTER 7

### COST ANALYSIS

Component	Cost
Raspberry Pi	Rs. 3200
Camera	Rs. 1670
SD card	Rs. 309
Wifi Module	Rs. 425
<b>Total:</b>	<b>Rs. 5604</b>

Table 7.1: Component and Total Cost of the Project

Amongst the projects analyzed while designing this project, the closest one is Elgin traffic management system. It is maintained by UK government, a community of drivers update the details about the emergency roads continuously. The significant difference in cost is primarily due to man work is reduced by lot. The camera cost may vary based on the position of the camera. If camera is placed far from the road, costs involve for capturing distant images. The above cost analysis doesn't include the costs of internet connection and power supply.

## **CHAPTER 8**

### **CONCLUSION**

#### **8.1 Project Summary**

The primary objective for the project was to design and develop a device capable of detecting on road vehicles and classify them based on type of vehicle. This was achieved by using the camera connected to the raspberry pi and the data collected in the past for training the classifier. The incorporated application can also count the vehicles and shows the current traffic conditions of particular area.

#### **8.2 Project Conclusion**

The detection and classification module with xml cascades trained was one of major contribution of this capstone project. Other significant contributions include the development of web application which can provide the data collected by the prototype device on to map module with greater visualization in real time. The future work includes implementation of the project at multiple places which can provide the details about traffic all along the road. The web application can be improved by analyzing the past traffic data and provide graphical data to predict the future patterns. The future developments can include developing a mobile app with SMS service and provide much more features in the web application.



## CHAPTER 9

### REFERENCES

- [1] Raspberry pi. [Online]. Available: <http://www.raspberrypi.org/>
- [2] "Windows 10 for IoT". Raspberry Pi Foundation. 30 April 2015.
- [3] Oliveira, M.; Santos, V. Automatic Detection of Cars in Real Roads using Haar-like Features
- [4] Raspberry pi guide [Online]. Available: <http://www.raspberrypi.org/quick-start-guide>
- [5] History and Hardware [Online]. Available: [http://en.wikipedia.org/wiki/Raspberry\\_Pi](http://en.wikipedia.org/wiki/Raspberry_Pi)
- [6] Vehicle Detection, Tracking and Counting [Online]. Available: <https://www.behance.net/gallery/Vehicle-Detection-Tracking-and-Counting/4057777>
- [7] Haar cascade detection [Online]. Available: [https://github.com/andrewssobral/vehicle\\_detection\\_haarcascades](https://github.com/andrewssobral/vehicle_detection_haarcascades)
- [8] Training OpenCV classifier [Online]. Available: <http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html>
- [9] Haar training [Online]. Available: <http://note.sonots.com/SciSoftware/haartraining.html>
- [10] Advantages-Disadvantages [Online]. Available: <http://sites.google.com/site/mis237group/projectraspberrypi/home/what-is-raspberry-pi/pros-and-cons-of-the-raspberry-pi>
- [11] Arduino-vs-raspberry-pi [Online]. Available: <https://www.maketecheasier.com/arduino-vs-raspberry-pi/>
- [12] Schematics for Raspberry Pi [Online]. Available: <https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/README.md>
- [13] Related Project: <https://code.google.com/archive/p/opencv-lane-vehicle-track/>
- [14] Opencv. [Online]. Available: <http://opencv.org/>
- [15] Cascade classification [Online]. Available: [http://docs.opencv.org/2.4/modules/objdetect/doc/cascade\\_classification.html](http://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html)
- [16] DVDVideoSoft. [Online]. Available: <http://dvdvideosoft.com/>

- [17] Virtualdub. [Online]. Available: [www.virtualdub.org/](http://www.virtualdub.org/)
- [18] Django Tutorial. [Online]. Available:  
<https://docs.djangoproject.com/en/1.9/intro/tutorial01/>
- [19] Haar Cascade Tutorial [Online]. Available:  
<https://www.youtube.com/watch?v=KFC6jxBKtBQ>
- [20] Mahdi Rezaei, (2015), Creating a Cascade of Haar-Like Classifiers.
- [21] Mapbox API [Online]. Available: <https://www.mapbox.com/api-documentation/#introduction>
- [22] Django. [Online]. Available: <https://www.djangoproject.com/>
- [23] Paul Viola and Michael J. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. IEEE CVPR, 2001.
- [24] Python. [Online]. Available: <https://www.python.org/>
- [25] SQLite Python Tutorial [Online]. Available:  
[www.tutorialspoint.com/sqlite/sqlite\\_python.htm](http://www.tutorialspoint.com/sqlite/sqlite_python.htm)



# APPENDIX A

## SCHEMATICS

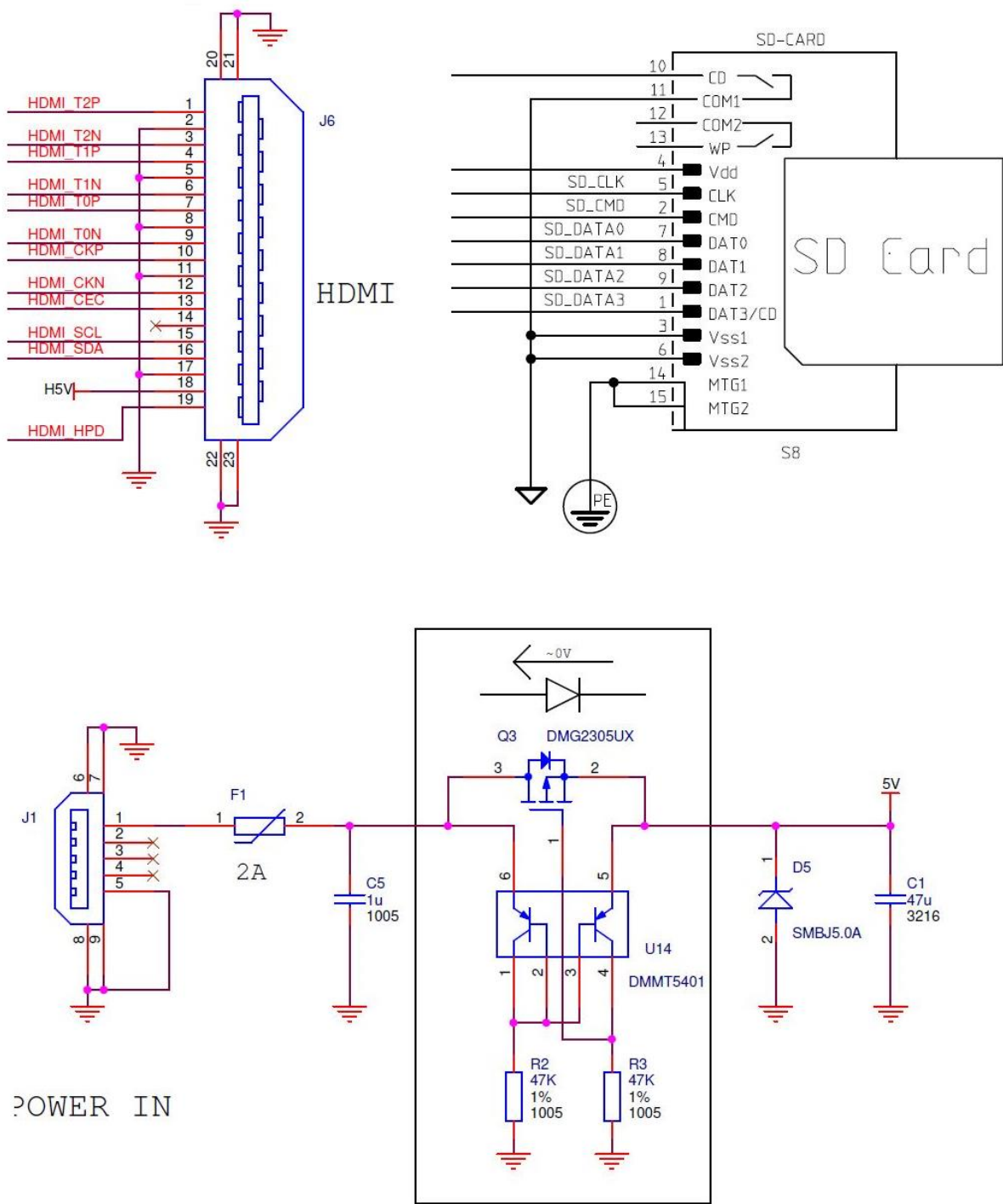


Fig. 10.1 Raspberry Pi Schematic

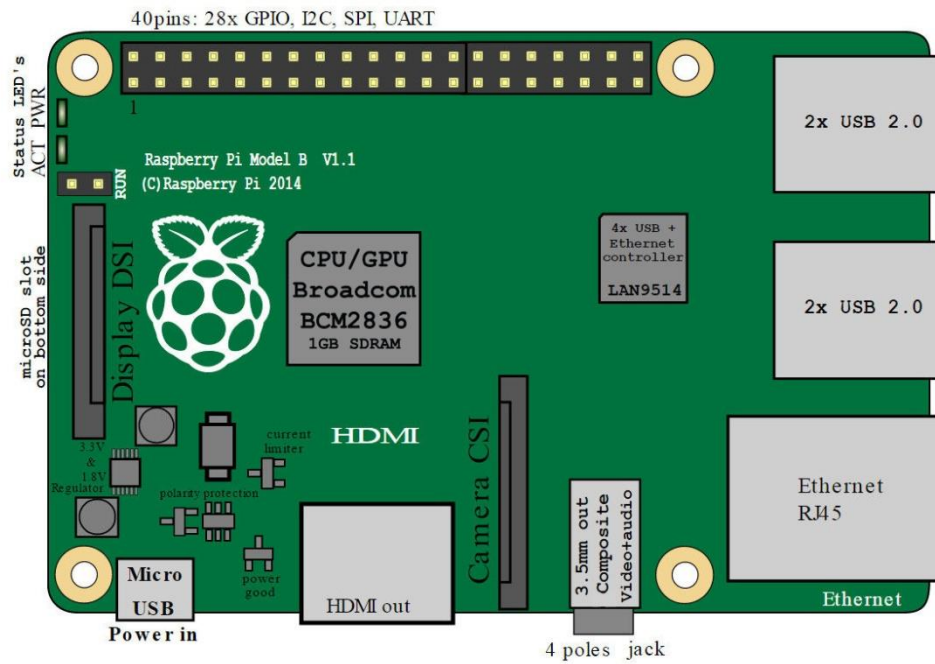


Fig. 10.2 Raspberry Pi B board

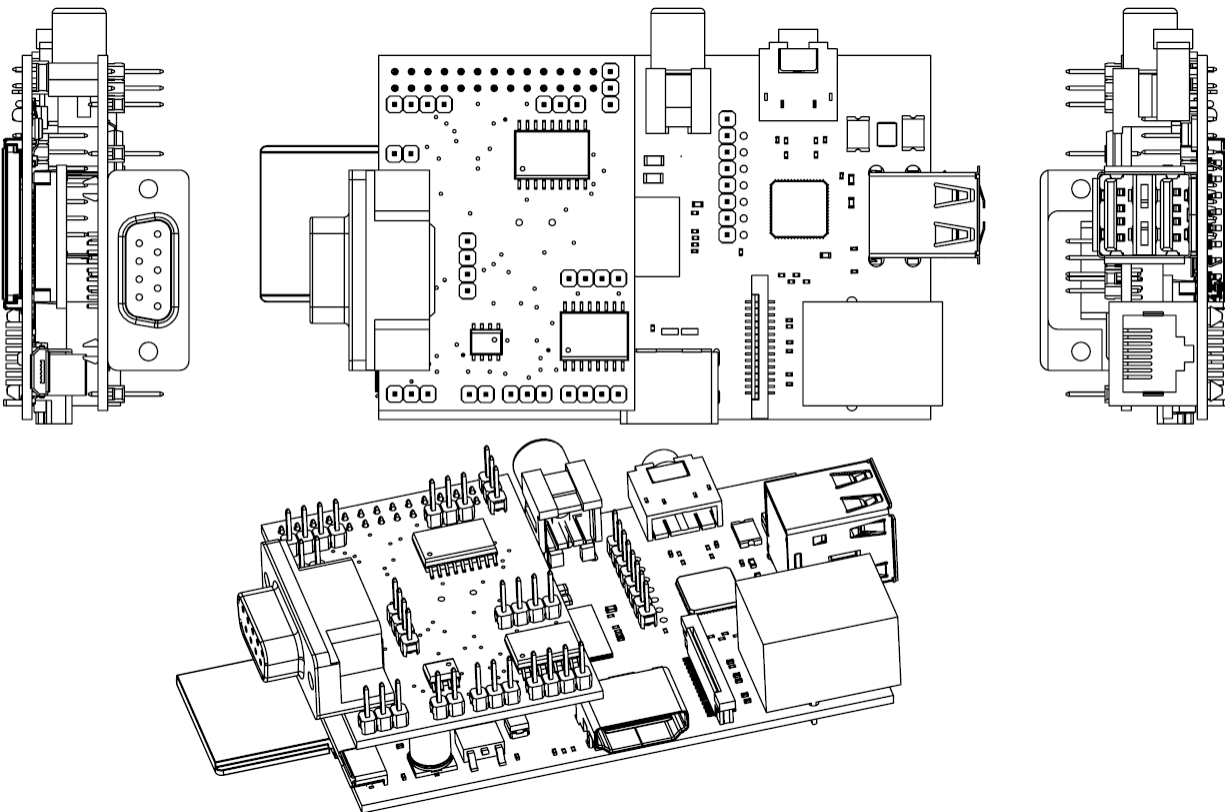


Fig. 10.3 Raspberry Pi Schematic



Fig. 10.4 Logitech c260



Fig. 10.5 Samsung 32GB micro SDHC

## **APPENDIX B**

### **REQUIREMENTS**

#### **Hardware Specification**

Raspberry Pi 2, Model B

Logitech C260

SanDisk 8GB sdhc card

Software Specifications

Python 3.5.1

Dvdvideosoft 6.3.10

Virtualdub 1.10.4

OpenCV 3.0

Windows 10 IOT core OS

#### **Web Application**

SQLite database

Django 1.9.4

Mapbox API v2.4.0

## APPENDIX C

### SOURCE CODE

#### Haarclassification.py

```
#Importing libraries
import cv2
#Importing xml files generated after training
car_cascade = cv2.CascadeClassifier('car-cascade.xml')
bike_cascade = cv2.CascadeClassifier('bike-cascade.xml')
for num in range(5,25) :
    #Reading images for testing
    img = cv2.imread('Random/pos'+str(num)+'.bmp')
    #Cropping images to focus only on vehicles
    img = img[80:444,0:535]
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #command to classify vehicles
    bike =
bike_cascade.detectMultiScale(gray,1.04,100,minSize=(100,100),
maxSize=(170,170))
    car = car_cascade.detectMultiScale(gray,1.07,50,minSize=(100,100),
maxSize=(170,170))
    #Code for classifying bike
    for (x,y,w,h) in bike:
        print (num)
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        cv2.imshow('img',img)
        #Waiting for user to move to the next image
        cv2.waitKey(0)
    #Code for classifying car
    for (x,y,w,h) in car:
```

```

print (num)
cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow('img',img)
cv2.waitKey(0)

```

### **Haarcounting.py**

```

#Importing libraries
import cv2
#Importing xml files generated after training
car_cascade = cv2.CascadeClassifier('car-cascade.xml')
bike_cascade = cv2.CascadeClassifier('bike-cascade.xml')
#Reading a video file
#Instead of path we can access webcam by putting 0.
vc = cv2.VideoCapture('output.avi')
bikeCount = 0
carCount = 0
carFlag = 0
bikeFlag = 0
while True :
    #only scan the vehicle once
    carFlagCheck = 0
    bikeFlagCheck = 0
    ret, img = vc.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #Command for classifying vehicles
    bike =
bike_cascade.detectMultiScale(gray,1.04,200,minSize=(100,100),
maxSize=(170,170))

```

```

car =
car_cascade.detectMultiScale(gray,1.04,130,minSize=(100,100),
maxSize=(170,170))
#classify vehicles before line
cv2.line(img,(100,230),(500,230),(255,255,0),3)
for (x,y,w,h) in bike:
    bikeFlagCheck = 1
    top = y
    if top > 150 :
        if bikeFlag == 0 :
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            bikeCount = bikeCount + len(bike)
            print ("Bike Count " , bikeCount)
            #It takes 2sec for a bike to cross the gate
            bikeFlag = 145
            break
        else :
            bikeFlag = bikeFlag - 1
            break
for (x,y,w,h) in car :
    carFlagCheck = 1
    top = y
    if top > 150 :
        if carFlag == 0 :
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
            carCount = carCount + len(car)
            print ("Car Count " , carCount)
            #It takes 3sec for a bike to cross the gate
            carFlag = 200
            break
        else :

```

```

        carFlag = carFlag - 1
        break
    `    #If both vehicles are not detected bikeFlag and checkFlag are
decremented
        if bikeFlagCheck == 0 and bikeFlag != 0 :
            bikeFlag = bikeFlag - 1

        if carFlagCheck == 0 and carFlag != 0 :
            carFlag = carFlag - 1

        cv2.imshow('img',img)
        k = cv2.waitKey(1)
cv2.destroyAllWindows()
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Webcamtovideo.py

```

#Importing libraries
import cv2

```

```

cap = cv2.VideoCapture(1)
#Video standard
fourcc = cv2.VideoWriter_fourcc(*'XVID')
#Provides path and window size
out = cv2.VideoWriter('OutputCam.avi',fourcc,20.0,(640,480))
while True :
    #Read camera from webcam
    ret , frame = cap.read()
    #Writing to a video
    out.write(frame)
    cv2.imshow('frame',frame)

```



```

    cv2.waitKey(1)
    #if q is pressed the video is stopped
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
out.release()
cv2.destroyAllWindows()

```

## WEB Module

### Maps/settings.py

```

"""Django settings for maps project.

Generated by 'django-admin startproject' using Django 1.9.4.

For more information on this file, see
https://docs.djangoproject.com/en/1.9/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.9/ref/settings/

"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR,...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/1.9/howto/deployment/checklist

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '$m$ddrz*up9s&_sungu2b3)7!xgyaqs*ozhal*%=55)z0l85h&'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

```

```

INSTALLED_APPS = [
    'markers.apps.MarkersConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

MIDDLEWARE_CLASSES = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'maps.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ]
        }
    }
]

```

```

],},},]
WSGI_APPLICATION = 'maps.wsgi.application'
# Database
# https://docs.djangoproject.com/en/1.9/ref/settings/#databases
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
# Password validation
# https://docs.djangoproject.com/en/1.9/ref/settings/#auth-password-validators
AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValida
tor',},
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',},
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',},
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',},]

# Internationalization
# https://docs.djangoproject.com/en/1.9/topics/i18n/

LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True

```

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.9/howto/static-files/
STATIC_URL = '/static/'
```

## Maps/urls.py

```
"""maps URL Configuration
```

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/1.9/topics/http/urls/>

Examples:

Function views

1. Add an import: `from my_app import views`
2. Add a URL to `urlpatterns`: `url(r'^$', views.home, name='home')`

Class-based views

1. Add an import: `from other_app.views import Home`
2. Add a URL to `urlpatterns`: `url(r'^$', Home.as_view(), name='home')`

Including another URLconf

1. Import the `include()` function: `from django.conf.urls import url, include`
  2. Add a URL to `urlpatterns`: `url(r'^blog/', include('blog.urls'))`
- ```
"""
```

```
#Importing required libraries
```

```
from django.conf.urls import include,url
```

```
from django.contrib import admin
```

```
from django.conf import settings
```

```
from django.conf.urls.static import static
```

```
#Mapping url to a particular regular expression
```

```
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^maps/',include('markers.urls'))
```

```
]#Setting static files to be displayed  
+ static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

### **Maps/manage.py**

```
#File required for running various commands in django  
#!/usr/bin/env python  
import os  
import sys  
if __name__ == "__main__":  
    os.environ.setdefault("DJANGO_SETTINGS_MODULE", "maps.settings")  
    from django.core.management import execute_from_command_line  
    execute_from_command_line(sys.argv)
```

### **Markers/admin.py**

```
from django.contrib import admin  
# Register your models here.  
from .models import Vehicle  
admin.site.register(Vehicle)
```

### **Markers/apps.py**

```
#To be included in installed apps in settings.py  
from django.apps import AppConfig  
class MarkersConfig(AppConfig):  
    name = 'markers'
```

### **Markers/views.py**

```
from django.db import models  
# Create your models here.  
#Vehicle model with required fields  
class Vehicle(models.Model):  
    vehicle=models.CharField(max_length=10)
```

```

        wheeler=models.CharField(max_length=5)
        lat=models.DecimalField(max_digits=9, decimal_places=7,
blank=True, null=True)
        lng=models.DecimalField(max_digits=9, decimal_places=7,
blank=True, null=True)
# Built-in function to compute the "informal" string representation
of an object.
    def __str__(self):
        return self.vehicle + ' ' + self.wheeler

```

### **Markers/urls.py**

```

from django.conf.urls import url
from django.conf import settings
from django.conf.urls.static import static
from . import views
#Mapping inside markers app in Maps project
urlpatterns=[
url(r'^$',views.index,name='index')
]+ static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

```

### **Markers/views.py**

```

from django.shortcuts import render
from .models import Vehicle
# Create your views here.
def index(request):
    veh_all=Vehicle.objects.all()
    context={
        'veh_all':veh_all}
#Rendering a template
    return render(request,'markers/index.html',context)

```

### **Migrations/0001\_initial.py**

```
# -*- coding: utf-8 -*-
# Generated by Django 1.9.4 on 2016-04-05 07:00
#Linking model to database
from __future__ import unicode_literals
from django.db import migrations, models
class Migration(migrations.Migration):
    initial = True
    dependencies = []
    operations = [
        migrations.CreateModel(
            name='Vehicle',
            fields=[
                ('id', models.AutoField(auto_created=True,
primary_key=True, serialize=False, verbose_name='ID')),
                ('vehicle', models.CharField(max_length=10)),
                ('wheeler', models.CharField(max_length=5)),
                ('lat', models.DecimalField(blank=True,
decimal_places=7, max_digits=9, null=True)),
                ('lng', models.DecimalField(blank=True,
decimal_places=7, max_digits=9, null=True)),],),]
```

### **Markers/index.html**

```
<!DOCTYPE html>
<html>    <head>
<meta charset=utf-8 />
<title>A simple map</title>
<meta name='viewport' content='initial-scale=1,maximum-scale=1,user-
scalable=no' />
<script
src='https://api.mapbox.com/mapbox.js/v2.4.0/mapbox.js'></script>
```

```

<link href='https://api.mapbox.com/mapbox.js/v2.4.0/mapbox.css'
rel='stylesheet' />
<style>
  body {          margin:0; padding:0;
                }

  #map {          position:absolute; top:0; bottom:0; width:100%;
                }

  #legend{        border : 2px solid white;
                  position:absolute;
                  top:120px; right:40px; background : white;
                }
</style> </head>
<body>
<div id='map'></div>  <script>
L.mapbox.accessToken =
'pk.eyJ1IjoidmlwdWwyNSIsImEiOiIjajW13eGd3b3AwM2YwdXdtNG45M2NpdGR0In0.dd
SQPg_l0f6H2mNDpzzh0w';
var map = L.mapbox.map('map', 'mapbox.streets')
    .setView([12.841077, 80.153179], 17);

var route = { type: 'LineString', coordinates: [] };
var route1 = { type: 'LineString', coordinates: [] };

var start = [12.842264, 80.152833];
//Polyline path for vehicles to move on a map
var lat =
[12.842396,12.842323,12.842323,12.842218,12.842134,12.842071,12.842019
,12.841977,
12.841904,12.841810,12.841747,12.841674,12.841591,12.841486,12.841423,
12.841329,12.840984, 12.840963, 12.840942, 12.840918, 12.840894,
12.840863, 12.840816, 12.840785,12.840746, 12.840704, 12.840671,

```



```

12.840637, 12.840608,12.840579, 12.840548,12.840519,
12.840477,12.840438, 12.840386];
var lng =
[80.152772,80.152793,80.152825,80.152868,80.152954,80.153040,80.153115
,80.153179,
80.153243,80.153297,80.153394,80.153469,80.153533,80.153586,80.153597,
80.153599,80.153496 , 80.153483,
80.153472,80.153453,80.153437,80.153418,80.153388,80.153364,80.153340,
80.153305,80.153281,80.153235,80.153197,80.153157,80.153119,80.153086,
80.153038,80.152980,80.152921];
for (var i = 0; i < 35; i++) {
    start[1] = lat[i];
    start[0] = lng[i];
    //For pushing the lat and lng to route variable
    route.coordinates.push(start.slice());
}
lat1 = new Array();
lng1 = new Array();
lat1[0] = 12.842396;
lng1[0] = 80.152772;
for ( i = 1; i < 10; i++)
{
    lat1[i] = lat[i-1] + 0.0004 ;
    lng1[i] = lng[i-1] + 0.0004;
    route1.coordinates.push(start.slice());
}
//Polyline creation
var poly={
    color:'#3b2dca',
    weight:8,
    fillOpacity: 0.6

```

```

};
// Add this generated geojson object to the map.
L.geoJson(route1,poly).addTo(map);
var j = 34;
k = 0;
lat = 12.840386;
lng = 80.152921;
bike = 0 ;
car = 0;
finallat = 12.842396;
finallng = 80.152772;
count = 0;
bikecount = 0;
carcount = 0;
marker = new Array();
mark1 = new Array();
vech = new Array();
//Reading variables from django database
{% for veh in veh_all %}
{% if veh.vehicle == 'car' %}
car = car + 1;
{% else %}
bike = bike + 1;
{% endif %}
{% endfor %}

for(i=0;i<car;i++)
{
    //Designing markers for car
    marker[k] = L.marker([lat,lng], {
        icon: L.mapbox.marker.icon({

```

```

        'marker-color': '#f86767',
        'marker-symbol': 'car',
        'marker-size': 'large',
    }));
    lat = lat + 0.00001;
    lng = lng + 0.00001;
    vech[k] = 'car';
    k++;}

for(i=0;i<bike;i++)
{
//Designing markers for bike
    marker[k] = L.marker([lat,lng], {
        icon: L.mapbox.marker.icon({
            'marker-color': '#abdca7',
            'marker-symbol': 'bicycle',
            'marker-size': 'large',
        }));
    lat = lat + 0.00002;
    lng = lng + 0.00002;
    vech[k] = 'bike';
    k++;}

for(i=0;i<k;i++)
{
    //Adding markers assigned to car and bike into the map
    L.geoJson(route,poly).addTo(map);
    marker[i].addTo(map);}
tick(marker,vech);

//Function to show animation of markers
function tick(mark,vehicle)

```

```

{   if ( count == k)
    {   for ( clear = 0 ; clear < count ; clear++ )
        {           //map.removeLayer(mark1[clear]);
        }
        return
    }

    mark[count].setLatLng(L.latLng(
    route.coordinates[j][1],
    route.coordinates[j][0]));
    if (--j > 0)
        {   //Setting the speed of vehicle
            setTimeout(function(){tick(mark,vehicle)}, 125)
        }
    if( j == 0)
        {   map.removeLayer(mark[count]);
            j = 34;
            if (vehicle[count] == 'car')
            {
                carcount += 1;
                document.getElementById('car').innerHTML = carcount;
                mark1[count]= L.marker([finallat, finallng], {
                    icon: L.mapbox.marker.icon({
                        'marker-color': '#f86767',
                        'marker-symbol':'car',
                        'marker-size':'large',})
                }).addTo(map);}
            if (vehicle[count] == 'bike')
            {   bikecount += 1;
                document.getElementById('bike').innerHTML = bikecount;
                mark1[count] = L.marker([finallat, finallng], {
                    icon: L.mapbox.marker.icon({

```

```

        'marker-color': '#abdca7',
        'marker-symbol': 'star',
        'marker-size': 'large',
    })
    }).addTo(map);    }

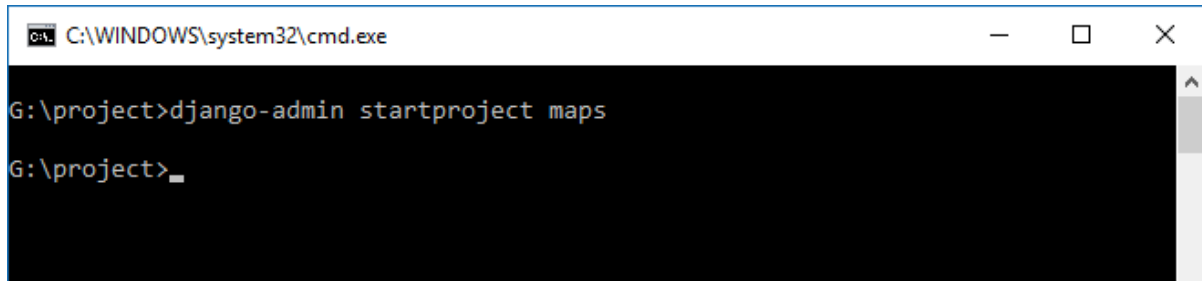
    finallat = finallat + .00002;
    finallng = finallng + .00001;
    count++;
    tick(mark,vehicle)
    }}
</script>
<div id='legend' >
    <strong>Vehicle Count</strong>
    <br>
    <!-- To load images onto the legend -->
    {% load staticfiles %}
    
    <label id = "bike">0</label>
    <br>
    {% load staticfiles %}
    
    <label id = "car">0</label>
    <br>
</div>
<script>
    //map.legendControl.addLegend(document.getElementById('legend').i
nnerHTML);
</script>
</body>
</html>

```

## APPENDIX D

### INSTRUCTIONS

#### Django Screenshots

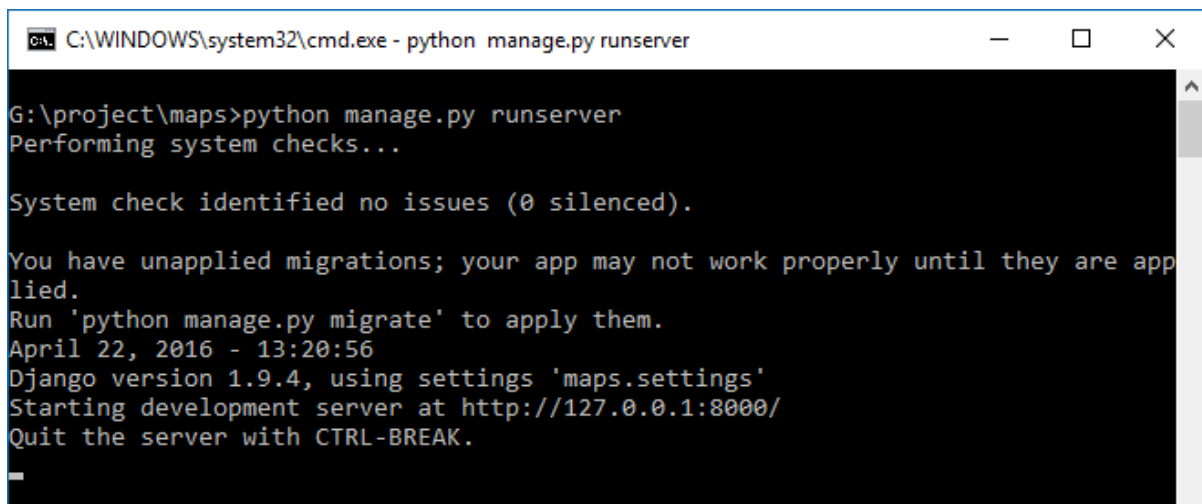


```
C:\WINDOWS\system32\cmd.exe

G:\project>django-admin startproject maps

G:\project>_
```

Fig. 1.Creating a project



```
C:\WINDOWS\system32\cmd.exe - python manage.py runserver

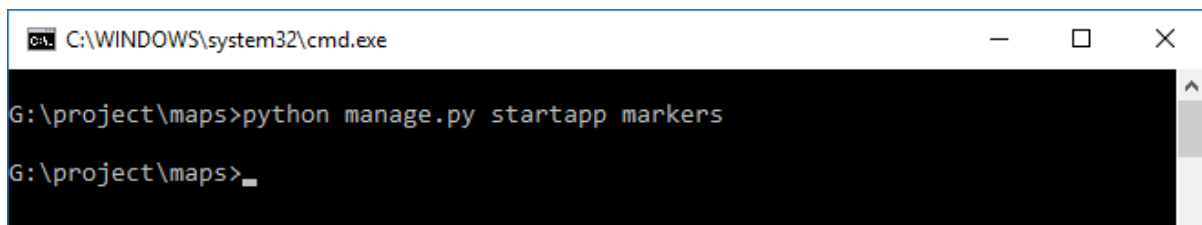
G:\project\maps>python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.
Run 'python manage.py migrate' to apply them.
April 22, 2016 - 13:20:56
Django version 1.9.4, using settings 'maps.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

_
```

Fig. 2.Start server

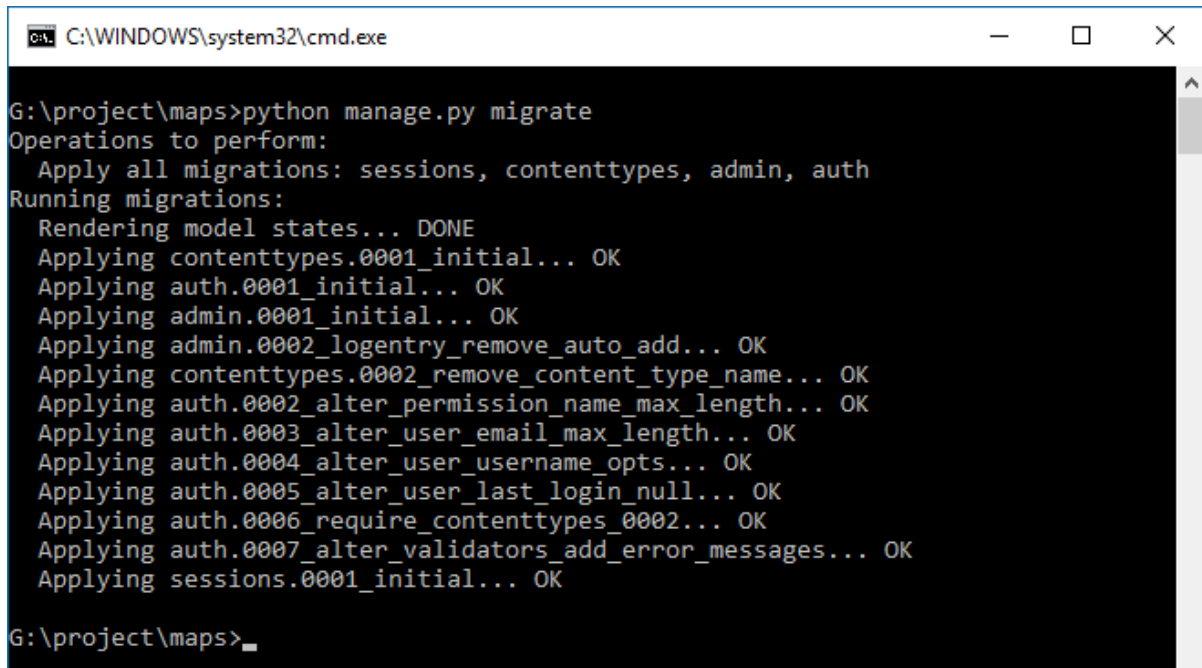


```
C:\WINDOWS\system32\cmd.exe

G:\project\maps>python manage.py startapp markers

G:\project\maps>_
```

Fig. 3.Create an app

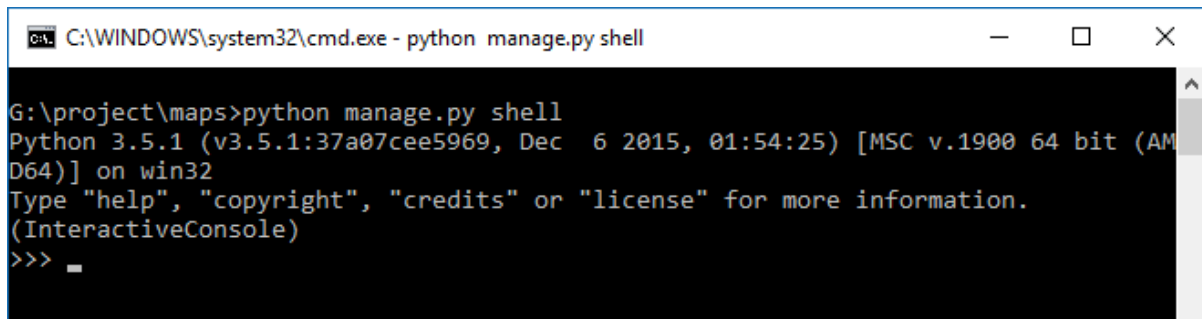


```
C:\WINDOWS\system32\cmd.exe

G:\project\maps>python manage.py migrate
Operations to perform:
  Apply all migrations: sessions, contenttypes, admin, auth
Running migrations:
  Rendering model states... DONE
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying sessions.0001_initial... OK

G:\project\maps>_
```

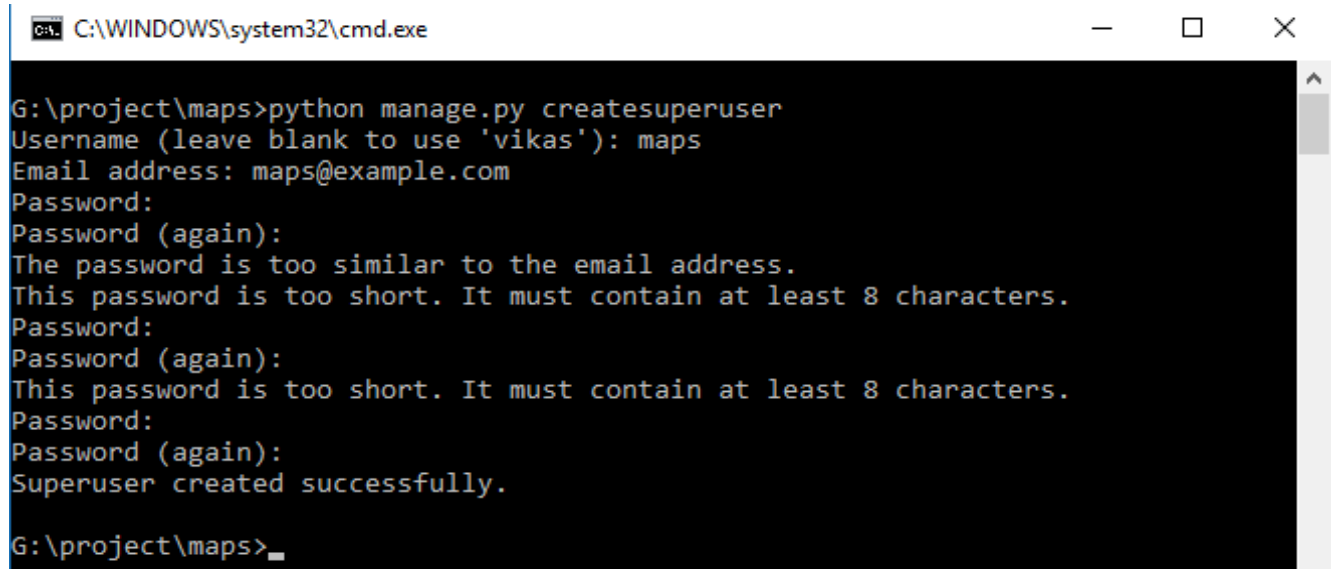
Fig. 4.Migrating required databases



```
C:\WINDOWS\system32\cmd.exe - python manage.py shell

G:\project\maps>python manage.py shell
Python 3.5.1 (v3.5.1:37a07cee5969, Dec  6 2015, 01:54:25) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>> _
```

Fig. 5.Interactive shell console



```
C:\WINDOWS\system32\cmd.exe

G:\project\maps>python manage.py createsuperuser
Username (leave blank to use 'vikas'): maps
Email address: maps@example.com
Password:
Password (again):
The password is too similar to the email address.
This password is too short. It must contain at least 8 characters.
Password:
Password (again):
This password is too short. It must contain at least 8 characters.
Password:
Password (again):
Superuser created successfully.

G:\project\maps>_
```

Fig. 6.Creating a superuser