Traffic Condition Awareness using Kalman Filter Technique with the Aid of Arduino and Matlab Embedded System

Author
Samir Karim, Nazmus Sa-Adat, Soumik Shadman, Mashrur Hasnaen

A thesis submitted in fulfillment of the requirements for the degree of B.Sc. in Electrical and Electronic Engineering

Supervisor
Dr. A.K.M Abdul Malek Azad Professor, EEE Department

Department of Electrical and Electronic Engineering,
BRAC University

August, 2018

Table Of Contents

DECLARATION

This is a submission to the Department of Electrical and Electronic Engineering for the purpose of obtaining a Bachelor's degree Bachelor of Science in Eletrical and Electronic Engineering. We, hereby declare that the results of this research are solely dependent on our research. Credits for any resources used are provided in the reference section. This paper was not used for any other academic or non-academic purpose and was not submitted for any other degree.

Signature of Supervisor: ____________________

Dr. Azad AKM

Professor

Department of Electrical and Electronic Engineering

BRAC University

Signature of Authors:

1. _________________________

Samir Karim

2. _________________________

Soumik Shadman

3. _________________________

Nazmus Sa-Adat

4. _________________________

Mashrur Hasnaen

List of Figures

List of Tables

Abbreviation

ACC – Actual Car Count
KPCC – Kalman Predicted Car Count
JSON – JavaScript Object Notation
LCD – Liquid Crystal Display
LED – Light Emitting Diode
URL – Universal Resource Locator
API – Application Programming Interfaces
REST API – Representational State Transfer Application Programming Interfaces
GSM – Global System for Mobile Communication
PHP – Hypertext Preprocessor
GPS – Global Positioning System
HTML – Hyper Text Markup Language
ARI – Accident Research Institute

Acknowledgements

We would like to thank our supervisor DR. Azad AKM for his continuous guidance and support for the completion of our thesis. We would like to acknowledge the The Control & Applications Research Centre (CARC) for motivation in completion of this project work. At last but not the least, we would like to thank our parents and family for their undying sacrifices so that we could finish this thesis without any problems.

Abstract

Traffic congestion in the city of Dhaka is reaching a new pinnacle every day. Owing to numerous private cars and local buses, the gridlocks in Dhaka city can be termed as one of the worst in the world. In a recent study done by Bangladesh University of Engineering and Technology(BUET) and Accident Research Institute (ARI), it was stated that around 5 million working hours are being lost every year in Dhaka city due to traffic gridlocks. This results in a loss of worth Tk.37,000 crore. In order to resolve this issue and help manage the gargantuan problem, we decided to tackle this by the implementation of Kalman Filter in order to predict the level of congestion at a particular road according to time variance of the day. The reason for using Kalman Filter is because its accuracy. Other reasons being that it can be simulated using Matlab and can be constantly updated using the latest data.

Chapter 1
Introduction

In this chapter, we first introduce the topic of Kalman Filter being used to predict the density of vehicle across a certain road in variable time. Then, the main research objectives are outlined followed by the research contributions. Finally, a general overview of the thesis is presented.

1.1 Literature review and problem statement

In an attempt to serve humanity and contribute to the scientific community, the idea of our thesis came into existence. Traffic congestions are a nightmare that every city, big or small, face now and as it seems, will face in the future as well. We attempt to solve this problem by enabling the commuters to select their route with the least number of cars during the morning rush hour. We plan on doing so by using practically collected data and inserting it into the Kalman Filter algorithm. This algorithm would in turn, calculate out the future number of cars crossing a particular road in the rush hours. Furthermore, the calculated data will be used to calculate the traffic flow, vehicle density, headway and average spacing. After doing so, the calculated set of data will be uploaded onto a server. This set of data will then be displayed onto our custom made mobile application using a preset color code for the routes. All these ideas and work procedures were motivated and helped by rigorous literature review present in [1-10]. This way, proper utilization of roads and time can be achieved. Even though there are other online applications which tend to serve the same purpose, their usage of metadata from cell phones to show the traffic congestion of roads, raises a matter of personal security.

One of the main reasons why traffic jams occur is due to excess vehicles using the same route to reach its destination. We believe that proper distribution of vehicles across different routes will result in a fanning out of the vehicle density. In doing so, the load on a particular road can be minimized and the chances of gridlocks dwindle down. Even if the route selected with the help of our application prove to be lengthier than other routes, the fact that there will be a continuous healthy flow of traffic will ensure that commuters do not lose valuable time sitting idle. There have been previous research papers suggesting the use of kalman filter to predict the number of vehicles on a road but its proper utilization for problem solving were absent.

1.2 Motivation of Work

Living in a city with one of the world's highest population density is no easy task. Added with poor public transport facilities and improper traffic management system, a simple commute within the city can turn out to be a nightmare. This is where our idea of spreading the traffic load form selected roads to multiple routes came into existence. We wanted to tackle traffic congestion with an embedded system which would be easy to set up and effective. Upon reading numerous research papers on Kalman Filter being used to predict traffic congestions, an easy to use vehicle counter along with a proper implementation method was lacking. We thought, if commuters could plan their routes according to their desired time beforehand and let the route be known to a central system, then, our vision of a distributed traffic channel will come into a reality.

1.3 Research Objective

1. To have a clear concept of the Kalman Filter Algorithm.
2. Design and construct a real time vehicle counting device with Arduino.
3. Design a system which will upload collected data onto a server, download that data onto the Matlab software, perform calculations and then use the calculated values to perform further calculations to find out traffic flow and upload that onto a server which in turn would be used in our mobile application.
4. Develop a mobile application which would show our data using a color code.
5. Finally, evaluation of the overall performance of our traffic management system.

1.4 Contribution of the Research

The main contributions of this thesis are as follow:

1. Inexpensive process of vehicle counting
2. Easy to understand traffic flow prediction results.
3. A headway to further research on minimizing traffic congestion in and overpopulated Dhaka city.
4. An overall control system has been designed and implemented which might prove to have

practical use on a large scale.

1.5 Thesis Organization

The rest of the dissertation is organized as follows:

Chapter 2: *Kalman Filter Algorithm and Traffic Parameters*
In this chapter, the reason for selecting the Kalman Filter has been presented. For example, how it puts a clear view of functionality and its uses in the unique capability of future prediction. Furthermore, we discuss the different traffic parameters and why the parameter of traffic flow was selected to be implemented into our system. At the end of this chapter, brief discussions of further improvement of the Kalman Filter are done.

Chapter 3: *Air pressure vehicle counter using Arduino*
This whole chapter is based on the discussion of constructing an Arduino based air pressure vehicle counter system in details. Moreover, the reason why this inexpensive method can be implemented on a large scale has been presented in terms of advantages. The elements of the modules and sensor roles have been gone into in depth to have a clear understanding of this microcontoller counter system.

Chapter 4: *Data Flow Structure and implementation*
In this chapter, the concept of how the data, that is being collected by our vehicle counter, is being uploaded, downaloded, calculated upon and then finally uploaded again is presented. The data that is being sent by the counter are, number of vehicles crossing a particular point in five-minute interval, Longitude and Latitude of the counter and a time. Further into into the chapter we discuss how this set of data is being used to predict the traffic flow of a particular road.

Chapter 5: *Matlab Simulation and Calculation*
In this chapter simulation result of the data set is presented in terms of analytical context. In the last chapter, we have seen how the data is being collected and made available for Matlab to utilize and output a predicted data set. But here we focus on the work of how the Matlab software takes a data reading, calculates the data using the Kalman Filter to output a predicted traffic flow count to make the whole project effective. We will further analyze them in order to check the performance and justifications of those calculated data.

Chapter 6: *Data Analysis*

In this chapter, we discuss our calculations, error percentages. After comparing the data we came up with deductive decisions about how accurate our process was. Furthermore, we can see a pattern forming between the actual car count and kalman predicted car count, which is very encouraging and vital to establish that our research has been on track.

Chapter 7: *Android Mobile Application*

This chapter introduces the function of how an android mobile application can be used to represent our data in a graphical way for commuters to understand. The chapter will further discuss how the application will work on the back end to bring forward a meaningful representation of the data. Further improvements of the application are also discussed briefly.

Chapter 8: *Conclusions and Future Work*

In this last chapter the main results of this dissertation is summarized, and some concluding remarks and identify potential directions for future research has been given.

Chapter 02
Kalman Filter and Traffic Parameters

In this chapter, the reason for selecting the Kalman Filter has been presented. For example, how it puts a clear view of functionality and its uses in the unique capability of future prediction. Furthermore, we discuss the different traffic parameters and why the parameter of traffic flow was selected to be implemented into our system. At the end of this chapter, brief discussions of further improvement of the Kalman Filter are done.

2.1 KALMAN FILTER

The kalman filter is an Iterative process that uses a set of equations and consecutive data inputs to quickly estimate the true value, position, velocity, of any object being measured when the measured values contain unpredicted or random error or variation. In other words, A Kalman filter is an optimal estimator which means it infers parameters of interest from indirect, inaccurate and uncertain observations. It is a recursive process so that new measurements can be processed as they arrive time to time. The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

2.1.1 General purposes of Kalman Filter:

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft. Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter also works for modeling the central nervous system's control of movement. Considering the time delay between issuing machine commands and receiving sensory feedback, application of the Kalman filter always deals with a realistic model for estimating the current state of the machine system and issuing updated data commands.

The Kalman filter has always been a very impactful autonomous methodology when it comes to controlling data for noisy and multi variable input systems. The basic operating principle of a Kalman filter is:

Insert noisy data into the system ⇒ Taking much possible less noisy data out.

However, the general applications of a Kalman filter are numerous in different categories which are stated as follows:

• One major application is tracking objects like vehicles, missiles and also their speed which is in some case very complex one.

•Used in complex computing like fitting data of Bezier patches which are often noisy and not stationary data.

•Since its multi dimension approach is very useful for tracking of objects, it works as a powerful tool for navigation system as well.

•However, modern research has developed the Kalman filter to be used in numerous computer vision applications such as:

- Stabilization of depth measurements,
- Collection and analysis of data from radar.
- Improved laser scanner system and stereo camera such as FLIR Stereo Vision Camera for velocity and depth measurements.

-Cluster and feature tracking and so on.

2.1.2 Our work purpose and the kalman filter:

The kalman filter is best in use when it comes to work with multiple data system say as for a system if there are 50 to 100 inputs and we need to predict any estimation based on those inputs what we need to do is finding the average of those values and also see the variations of those data inputs and it might take a lot of time and of course the problems with the mathematical calculation through different logic is another big issue for all those data to be calculated. However, the kalman filter does that same thing very easily and quickly as it just works on some of those data inputs in the beginning

and so the variations in between those data inputs and through each iteration it improves the prediction for the true value with the least amount of error.
In our case the number of vehicles is of a massive data as the numbers are increasing day to day and so to track the numbers of vehicles and to generate a trustworthy outcome considering all the technical limitations and adverse traffic conditions our set up hardware system matches best to the algorithm of the kalman filter. Each time new data are put into the system and the data are updated time to time with proper calculations and several other parameters. Thus the Kalman filter algorithm is described as clearly as possible in the following section and shows how the whole processing of data is done.

2.1.3 Operating principle and basic algorithm of Kalman filter:

The basic algorithm is categorized into three sections, titled as; calculating the Kalman gain, calculating the current estimation and calculating the error in the current estimation. These three calculations are done one after another and it's an iterative process. Once the kalman gain is calculated it is then fed into the calculation of the current estimation and once both the kalman gain and the current estimation are calculated then the kalman filtering wroks on how much is the error in the new estimated or predicted value and the iteration ends with that and then the next iteration begins with all the previous estimated data and the calculation of the new data inputs and this whole process keeps on going unless we have the desired prediction based on the true value of the data we are working on. so, the three major areas named as the kalman gain or sometimes its ust the gain, the calculation of the current estimation and the calculation of the error in the current estimation. In the block diagram for the kalman filter explained here we see how the whole system is working based on these three majior areas.

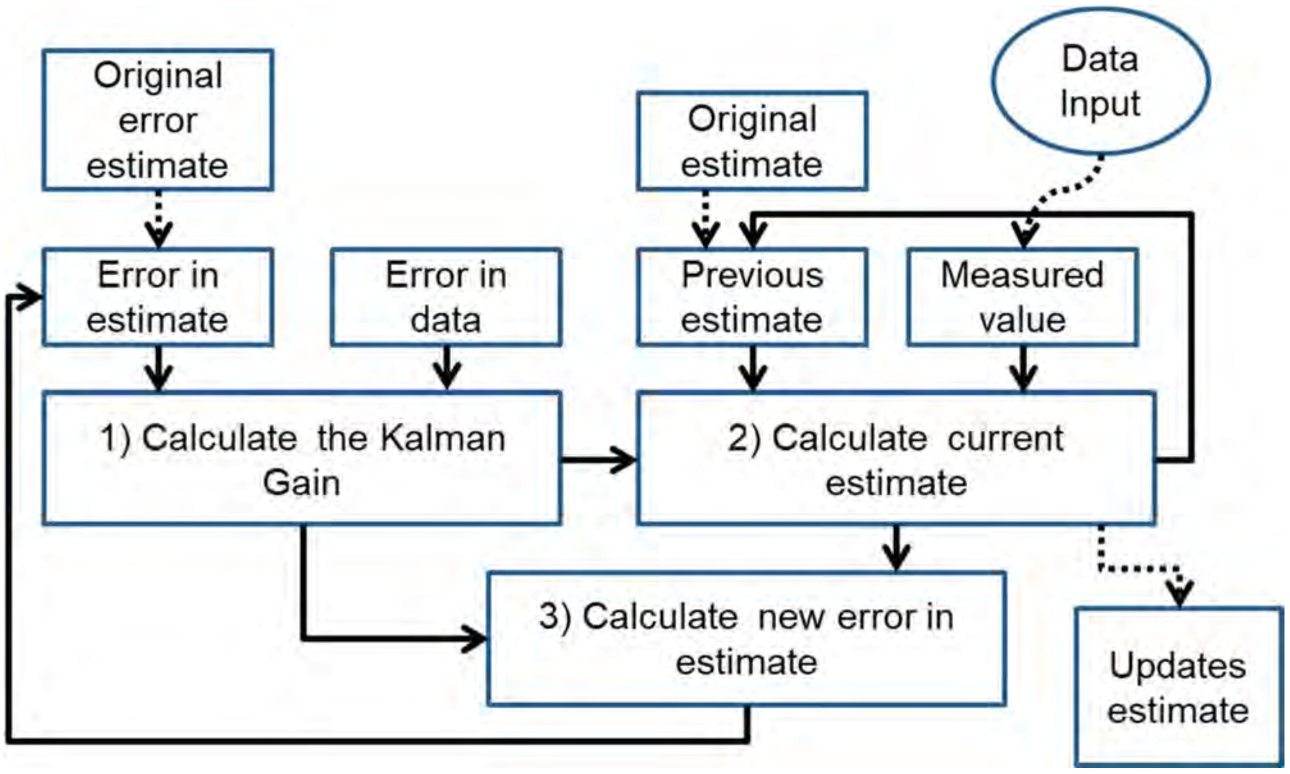


Diagram: 1 Kalman Filter Flow Chart

2.1.4 The Kalman Gain:

At first to calculate the kalman gain we need two things the error in the measurement and the error in the estimation. How and what are these two parameters relation in the kalman gain calculation is the bigger picture here. The error in the measurement is what we get from the difference of the sensor value from the aurdiono based counter we have used on the road track. On the other had, the estimated error value is what we estimate based on the senseor value once we have the measurement error of the sensor value. These two are then feed into the kalman gain equation to calculate the kal man gain. What is the significance of the kalman gain with respect to these two error values? The answer to this complex one is very easily explained when we analyze the kalman gain equation along wth the other two sets of equations. The kalman gain puts and relative importance to show much to take into consideration between those two parameters known as the error in the estimate and the error in the data. When the error in the estimate is low then we trust more on ours estimation and if the error in the data is low then we trust more on the measured values by the sensor and most importantly what role the kalman filter plays there is what the kalman gain equation is all about and we will talk about it later on.

2.1.5 The Current Estimation:

After calculating the Kalman gain the system algorithm demands the next major processing principle which is the calculation of the current estimation for which we have a specific equation that comes out be a success with the very two important set of data called previous estimate and the measured data value. These two major datas are feed into the equation which is explained further in this section along with the other two equations of the kalman gain and the error in the estimation. The kalman gain again here puts a weight on the relation between the values of the estimation and the sensor. For a higher value of kalman gain the two data inputs here becomes significant and for the lowest value of the kalman gain the data inputs are less likely to be taken into consideration. One major issue is how we consider the previous estimation when it's the begening of the whole process? Well, we have had a solution to that aswel.when it is the case where we have no sets of predicted data of the vehicles we first took some days raw data by the sensor and then we calculated the average of those data and it helped us predict our expected data when we started the whole system and eventually the results were reliable and mathematically accurate to predict for future analysis. In practical sense this the best possible solution to calculate those raw data for estimation based on the adverse condition of any developing countries like Bangladesh and especially for the undergrad thesis students. We had to face many situations regarding time and he raw data and specially the security issues while collecting those data and of course the attention of the mass crowd on the road. Despite all these, ous goal and the working principle led us to our desired outcome and yes we did have a good starting with ours estimation based on the very day to day traffic condition and so we are satisfied with the outcome we have. The most important thing is that the current estimated value is the final outcome or simply the predicted value what is our goal to calculate. However, this current estimation is with some error which we calculate at the end of each iteration to be used when the new sets of data inputs are on hand later on the next iteration.

2.1.6 The Error in the Current Estimation:

The third major calculation is about the error present in each calculation of current estimation and we do have a separate equation for that too which is explained in the next section. This error calculation tells how accurate the new predictions are being after each iteration and this error is then used for the next time when we calculate kalman gain and other parameters. Every time, we see from the equation that the error is less than the previous time error and that proves how the estimates are getting close to the desired true value after each iteration.

2.1.7 Our system and the Kalman Algorithm:

In this section we are going to talk about in details how those three equations are performed and how the outcomes relate to each other to give us what is desired. At first, let's start with the kalman gain calculation and the Kalman gain is calculated as: $K = P/(P+R)$, where P is the error in the estimation which is taken manually first as described in the previous section which is how and in which way we predicted that error for the first time and all the times after the first attempt. It is close to the error of the sensor value, in other words, the error in measurement, R. This R value is assumed by comparing the previous weeks' data for a particular day at a particular time. From the equation we see the Kalman gain is dependent on the errors of the predicted error value and the error of the actual value. The Kalman gain often ranges between 0 and 1, where lower values of kalman gain indicates that there are more error in the actual value R. For that, we emphasize on the estimated error values as we see there are more errors in the measured values and we don't want the measured values upset the whole calculation and hence we get a gain which prioritizes the estimated error value P. However, things are the exact opposite if we have higher values for the Kalman gain which is close to 1 or in some cases it is 1. When we have a high value of kalman gain that is close to 1 then we see the errors in the measurements are less and it takes longer time to reach in to the desired estimation since the error in the estimation is too high. However, if the kalman gain is small close to 0 then the error in the measured values are high and so we put more weight on the estimated datas which helps up to zoom into the true value very quickly and so it does not let the measured values upset the estimation that much. With this idea in short our main target is to focus on how to reach to value of the kalman gain close to 0 so that we have more accurate predicted data and we don't want the measured values upset the whole system in each iteration.

This Kalman gain is important in order to calculate the current estimation X and also the error in the current estimation which is illustrated further. After the calculation of the Kalman gain, the current estimation r is made through the following equation;

$$r = X + K*(Z-X),$$

Here, X is the value we estimate close to the values of previous two weeks (for the first time only). Once this process has started, this assumed value will be used as the previous estimated value of X. For the second time of this whole iterative process, the calculated value is used and this will keep on going till the end of iteration. For the first time the X value is calculated based on the average of the sensor data manually which is mentioned very clearly earlier. In the equation of calculating the current estimation r, the kalman gain value tells us how much weight to put on the differences between the estimated data X and the measured data Z. Once we have a higher value of kalman gain we see the difference (Z-X) is of huge significance for the estimation of the present value and it takes longer time to get to a trustable estimated value time to time. Whenever, the kalman gain is less eventually the difference (Z-X) is multiplied by a small portion and thus contributes less to the calculation of the current estimation and what this signifies is that we are very near to the desired estimation as time goes by and the errors in our estimation is getting less and then we don't allow the measured sensor value Z upset the estimation anymore. Throughout the whole iteration process this practice continues which eventually leads to a lower value of kalman gain with the least amount of error in the present estimation of the traffic condition. The major issues here has been the decrease of the kalman gain in every iteration so that all the other parameters calculated based on the kalman gain are more accurate. So in this equation, mainly, the Kalman gain dictates which value to choose from, either the actual data or the assumed data, depending on the errors.

Finally, the calculation of the error in the current estimation is performed, which is of paramount importance for the next iteration to get started with the calculation of the Kalman gain and so on. The error in the estimation is calculated by:

$$p = (1-K)*P$$

The above equation is simply the short outcome of the following equation:

$$\begin{aligned} p &= RP/(R+P) \\ &= P*\{1-(P/P+R)\} \\ &= P*(1-K) \qquad \text{[Note: } K = (P/P+R)\text{]} \end{aligned}$$

where we see every time the error in the estimation is reduced as the previous values are multiplied by a factor less than 1 and as the iteration goes on, the prediction is becoming more and more accurate which means less error in the estimation P and for that the kalman gain is very less with each time iteration and the measured values Z are more likely to be less dominating in the system which indicates the error in the measured values(R) are more [referred to equation $K = P/(P+R)$] and for that the error in the current estimation eventually becomes very small (close to 0) [referred to equation p = (1-K)*P] as the K is very close to 0 the (1-K) is considered as 1 and therefore the P is eventually close to the p value which is the main target as the errors after the iterations are almost similar or same (when it's the exact prediction). In the end of the whole process minimizes the error and brings forward a predicted value which is the sole purpose of this entire algorithm.

2.2 Traffic Parameters

In this chapter, the basic concepts of traffic flow is presented. To begin with this particular section first, we would draw the attention to the sole purpose of our thesis project which is the traffic condition awareness and its proper prediction. However, these simple words have very complex outcome which is why we are using parameters we found through research. Traffic engineering leads us to the analysis of behaviours of the day to day traffic and to design the facilities for a much advanced and smooth, safe and economical operation of traffic system. However, traffic flow, like the flow of water, has several parameters associated with it to make the whole system vivid. The traffic stream parameters provide information regarding the nature of traffic flow, average spacing, speed, vehicle or traffic density, headway etc. which helps the analyst in detecting any variation in flow characteristics. Understanding traffic behaviour requires a thorough knowledge of traffic stream parameters and their mutual relationships.

However, these specific parameters are chosen based on several research on what are the key things to make us understand the traffic condition and how they are implemented and their relation with our kalman algorithm where all other necessary data are readily available? Traffic flow demand parameters represent the demand on a transportation system. The comparison between this and the capacity determines a facility's operational performance. Traffic flow measurement has many uses in the planning, design, and operation of highway facilities. Planners use flow measurements to classify streets, identify traffic trends, study origin-destination patterns, and predict revenue. Designers use flow information to design facilities. Traffic engineers use flow measurements to analyze traffic operations, accidents and investigate improvements such as priority lanes. Therefor the following chosen parameters are mentioned and explained further with their uses in describing the traffic condition are presented below.

2.2.1) Traffic Flow

The traffic flow parameter is based on the two important data input which are the number of vehicle (r) (both the measured one from the sensor and the kalman predicted value) and the time range (t) which is stated as follows:

$$Traffic\,Flow = \; r/t$$

The purpose of this parameter is to get the idea in general that how the numbers are dominating the traffic congestion for a particular time and here as the time period is very short (t=5 minutes or 300seconds) so it is easy to understand the root of the major problem. Through this little time duration data we are able to keep a track on the condition and often we are able to predict the traffic condition according to our necessity. We don't need to go through a lot of data which takes too long to be stored or nothing like that sort of time consuming process. Every 5 minutes later the data are uploaded and we can have access to those data (both the kalman predicted car count value KPCC and the actual car count value ACC).

The traffic flow thus helps us to measure the vehicle number for a particular time unit. For example in the given table of the data sheet we found the usual numbers of cars on the road on Monday were 161 according to our sensor value and 158.23 (see Table 01 Number of cars passing through with respect to time) based on the kalman predicted car count value. For each case these data can be used as the value of r for the number of vehicles and as we already mentioned the time range has been for 5 minutes or 300 seconds. By using the formula for traffic flow the result is 32.2 for ACC and for KPCC its 31.646 (see Table 3 Traffic Parameters table with Actual Car Count and Kalman Predicted Car Count). From those tables data we are having the idea how the traffic flow is varied with respect to time and thus we are able to predict whenever we need.

2.2.2) Vehicle Density

The very fundamental traffic flow characteristics are traffic flow, speed, and traffic density. We already defined traffic flow as the number of vehicles (r) passing a point in a given time period (t). The second parameter is the Traffic density which we define as the number of vehicles (r) occupying a unit length of roadway at an instant in time. However, we will focus on the latter level but the transportation engineers can study these characteristics at the microscopic and macroscopic levels. The following formula is used to calculate the vehicle density:

$$Vehicle\ Density = \ r/l$$

Where r is the total number of cars in a road (both from ACC and KPCC) occupying a given length of the roadway and *l* denotes the length of the roadway we have performed our field work on. Once we have these two inputs we can find the Vehicle density through which it is easier to predict how dense the road way is and from our practical knowledge we can clearly understand for what range of value for any particular roadway we call it a dense one or not. For example in the table (Table 3 Traffic Parameters table with Actual Car Count and Kalman Predicted Car Count) we can see the density value placed on the table for both ACC and KPCC and both those values are reliable one and every time these two sets of data are having similar pattern. However, for both case it is even possible to calculate the speed of the vehicle if we follow the following equation:

$$\text{Speed} = \frac{Traffic\ flow}{Vehicle\ Density}$$

$$= (r/t)/(\ r/l)$$

$$= l/t \text{ [Note: Speed= length/time]}$$

The major issue of calculating the speed is to put a limit and bring a balance on the speed of the fast moving vehicles on the high way to avoid accident. However, it is important to compare the speed of multi type vehicles moving around us daily and for this we don't need any other special technologies like Tachometers for Engine Speed, Speedometers for Travelling Speed and Accelerometers for measuring Acceleration and Deceleration, Radar for Determining Vehicle Speed, Average Speed Computers 81 and so on. Rather, we can very easily calculate the speed using the above formula when we need based on the collected data inputs and be aware of the average speed the vehicles that are moving on the road way we are travelling at any time. If the future research are done on this project then the government won't need the help of much complex solution rather such problems can be faced with these little but effective techniques.

2.2.3) Average Spacing

After calculating all the above parameters it is often easier to predict and know the condition of the traffic better if we calculate the parameters called Average Spacing and Traffic Headway. The Average Spacing is the distance between successive vehicles in a traffic roadway which tells us how the cars are organised when on the roadways and this is very important to avoid accidents and is very helpful for a systematic and organised traffic on the roadway. The formula for calculating the Average Spacing is as follows:

$$Average\ Spacing = \ 1/Vehicle\ Density$$

Often we see in Bangladesh cars and busses do come very close to each other on the road and thus it’s very risky and might cause accident as well. However, if there is any standard of the Average Spacing that can be applied for the awareness of the people then these awareness can be of huge success in reducing the amount of misacts on the roads. So again, the collected data (both ACC and KPCC) can be used to determine the average spacing for a given amount of length of a roadway and that is what we have placed in table in the data sheet(See Table 3 Traffic Parameters table with Actual Car Count and Kalman Predicted Car Count).

2.2.4) Headway

Finally if we calculate the time space it takes for two successive cars to cross a particular road section which is namely the Traffic Headway meaning the corresponding time between two successive vehicles as they pass a given point on the roadway then it is much easier to know how frequent the vehicles are available for that particular point of the road and thus its obviously very useful information for the traffic condition. So the formula for calculating the Traffic Headway is:

$$\boldsymbol{Headway = 1/Traffic\ Flow}$$

With the help of all these parameters we get to know not only the vehicles number but also the condition of the road and so these four particular parameters are of huge importance and yes these simple methodology can be used to solve major issues like daily traffic congestion for developing countries like Bangladesh.

In our thesis project the most useful outcome of these four parameters have been displayed on the android mobile phone application where we set different parameters based on the calculation of these four points and the users will find how these parameters are used to display the outcome on the app. Therefore, to sum up, this specific section has paramount importance in the traffic condition and its takes our thesis research to a new dimension with all the other successful outcomes.

Chapter 03

Air pressure vehicle counter using Arduino

In order to begin our research, we need to count the number of vehicles pass through the road. Thus, we decided to build a vehicle counter which would serve as the primary data collector. There are too many procedures to count the number of vehicles such as RFID, Camera, piezoelectric sensor, pneumatic tube, inductive loop etc. For this we had to choose any one so that the counter can be efficient and budget friendly. Most of the counting process are too much costly and some of them are not possible to implement in our country. Hence considering all the problems we choose the pneumatic which is an Arduino based vehicle counter prototype. The initial idea was discovered by this Youtube tutorial [2]. There, the demonstrator built a road tube counter using an Arduino microcontroller. However, this counting process was for low density road area, so we decided to upgrade the algorithm to a more robust one. That is when we came across the open source code found in [3]. However, this code too needed to be tailored specifically for our purpose as it was for small width road and for medium speedy vehicles. We had to customize the code a lot as we wanted to count the vehicle number and also wanted to upload the vehicle number in every five minutes. For the 1st few weeks we didn't connect the gsm module and its driver arduino. We just connected the 1st arduino with the modification for high density number of vehicle's road. With it we counted the vehicles and stored it into our database. After that we modified our final code, this final code was for counting the vehicles and also to upload the numbers of vehicle in every five minutes. The customized code algorithm is further discussed in this chapter. The counter is built using two Arduino Nano microcontrollers along with a pressure sensor and GSM sim808 module.

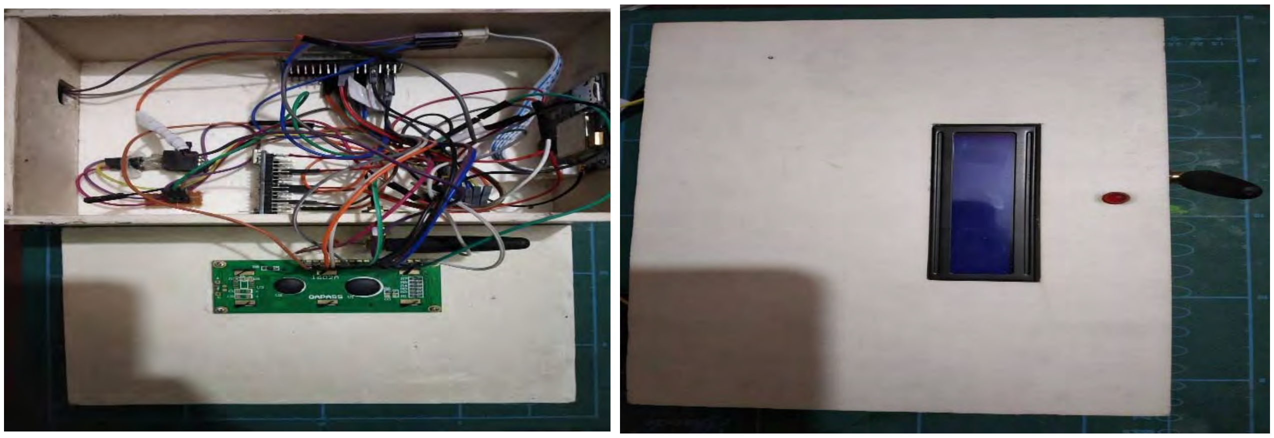


Diagram: 2 Counter Box

3.1 **List and Description of Components:**

1) Arduino Nano
2) Pressure Sensor (MPXV5050DP)
3) Rubber tube
4) Power Bank
5) LCD display
6) GSM module (SIM808)
7) LED

Arduino Nano:

This is a type of arduino which is small and board friendly. It has ATmega328p microcontroller. It is easy to use and can read most of the sensor analog and digital values. In our project we used it because it can read our pressure sensor's analog value and also can pass the data to the 2nd arduino so that it can drive the GSM module to upload the data. We also chose this so that it can be easily fit in our counter box.

Pressure Sensor (MPXV5050DP):

This is an integrated silicon pressure sensor which is actually the main part of counting the number of vehicles. By compression of air through the rubber tube the pressure sensor senses the air pressure and convert into electrical analog signal. These signal go to arduino nano which senses the analog signal and makes it usable.

Rubber Tube:

Silicon rubber rube which is used in our project for vehicle counting. One of the side is blocked by hot glue and another side is connected with the pressure sensor. When a vehicle pass through the rubber tube the air inside the air is compressed and the compressed air hits the pressure sensors nose.

Power bank:

This is used for powering up both the arduino. As our counter is a portable one so it needs to power up with portable battery or something else. We chose power bank as it gives a desire current supply with 5 volts and more usable.

LCD Display:

LCD is actually a liquid crystal display whose screen is an electronic display module with 16*2 segment. We used it so that we can see the vehicles number from the box. Though the numbers are sent to the database but we need to see this visually. It is easy to connect with arduinos, easy to fit in box and takes less power.

GSM module:

GSM is a modem of mobile communication, the full form is global system mobile communication. It can send text, make call, receive call, browse internet etc. we need to send the number of vehicle in our database so we used that module. We used SIM808 which is very small and can be easily fit in our counter box. It has an antenna. It receives and transmits data through arduino. It has also GPS mode which we did not use as it will consume much energy.

LED:

LED stands for light emitting diode. Here we used it to see either the sensor is taking the input or not. When the compressed air hits the sensor the LED blinks and we become sure that the sensor is working properly and counting the wheels to measure the vehicles number.

3.2 Experimental Setup:

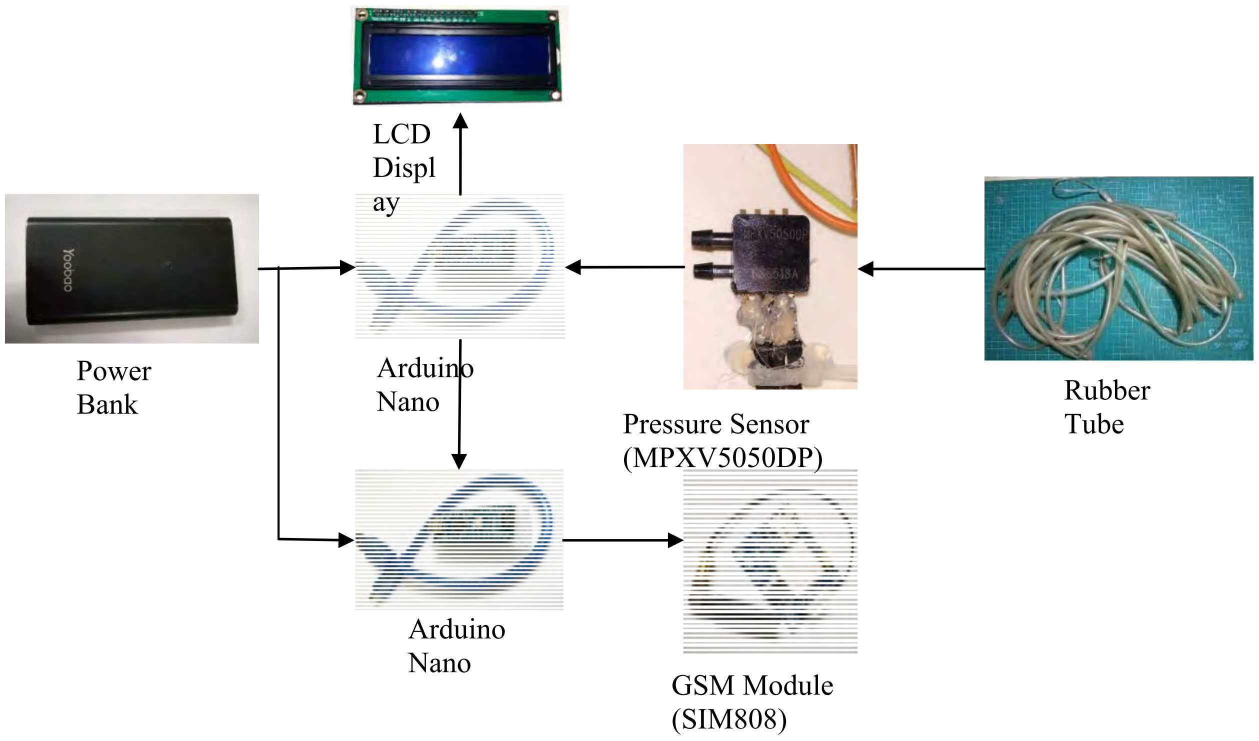


Figure 3: Experimental Setup of Components

Our initial though was that the first arduino counts the number of vehicles passing over the road tube with the aid of the pressure sensor. The change in air pressure inside the tube is sensed by the sensor and the arduino counts the number of wheels passing over it. The LED blinks and we get to know that the sensor is working. This data is then displayed using the LCD Display, the wheels number it counts and the number of vehicles. There, the first arduino supposed to connect with the pressure sensor, LED, LCD display and the second arduino, this second arduino with the GSM module. After getting the number of vehicles it will send the numbers to the second arduino which is the driver of the GSM module. We counted the number of wheels and the number of vehicles. With the aid of the arduino code which was modified from the source code [3], we measured the number of wheels and the number of vehicles. It was efficient and easy to count the numbers. However, there were some error, it missed a few number of vehicles.

This technique was enough but as we wanted to send the data to the database in order to make it real time. So we setup the connection of the GSM module and its driver arduino. In this process we used the first arduino to measure the number of wheels passes over the rubber tube. Thus the first arduino is connected with LED and second arduino. The second arduino with LCD display and GSM module. After getting the vehicle number, the data is sent to the second driver arduino which measures the number of vehicles from the total wheel count. After that it shows the number of vehicles in the LCD display. After every five minutes the arduino send the data to the GSM module. The GSM module takes the number from the arduino and hits the api URL to upload the data into the database. Before uploading the data, it takes some time to initialize the GSM module. The GSM module needs to be in data connect method otherwise it cannot send the data. We know there are multiple methods of uploading the data. These are GET, POST, HEAD and PUT etc. In our project we used the GET method to upload the data. The *php* code inside the website is also arranged in GET method. If the method does not match the arduino will never be able to upload the data to the server. The number of vehicles is sent with a particular digit. As we are not using the GPS part of the GSM module so this particular number going to represent a particular route.

3.3 Code explanation:

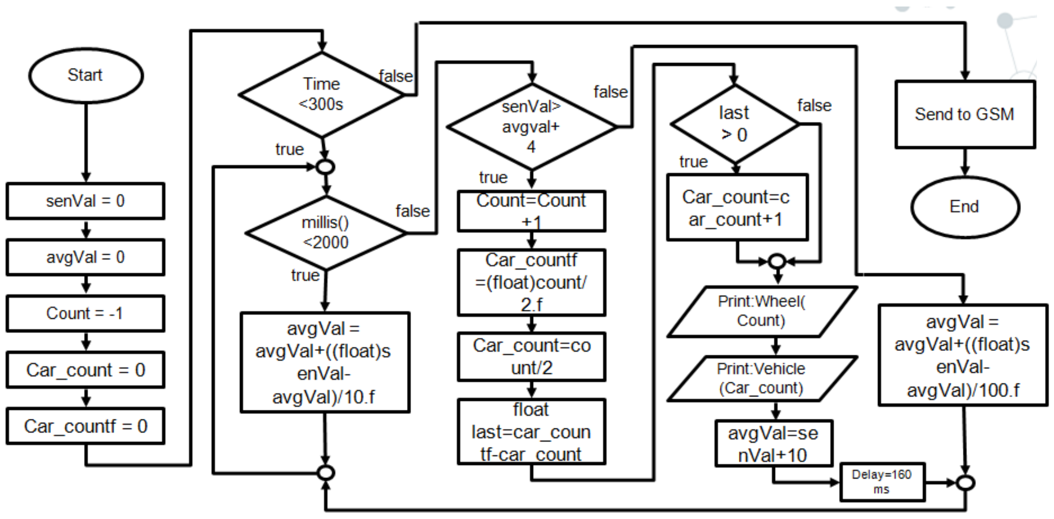


Figure 4: Arduino Code Flow Chart

The code is divided into two part. First one is for the 1st arduino which will count the number of wheels. In this code 1st we had to initialize the some values for the sensor. avgVal, senVal, Count, Car_count and Car_countf. ; senVal stands for the sensor value, avgVal is the average value of the sensor value, Count denotes the number of wheel count, Car_count is used for counting the vehicle number, Car_countf is the variable chosen to convert Car_count into a float number, float_last is the difference between car_count & car_countf. All started with 0 but the count started with -1, if count would start with 0 then it starts with 1 automatically for the 1st time. Then it enters in the 300s loop. As we are counting the number of vehicles for 5 minutes and 5 minutes = 300s so we took the loop. If the time is more than 300s it will send the last numbers of vehicle to the driver 2nd arduino so that the data can be upload onto our server using GSM module. While the time is within these 300 seconds, the variable millis returns the number of milliseconds since the Arduino board began running the current program. The purpose of the millis function is to increase the accuracy of the whole system. This function does not interrupt the entire process of the Arduino. Rather, it keeps on updating the average sensor value in the beginning of the process while the sensor value is being recorded. If the sensor value is greater than the average value, the Arduino increases the number of car count by 1. If false, the average value is modified in order to better the readings for the next air pressure difference. The delay of 160ms is used for the Arduino to function properly. The modifications done to the code from [2] is adjusting the delay between the first set of wheels and the second set of wheels. We have set a delay of 160ms based on practical results. More so, we have eliminated the problem that we faced when the rear wheel of the leading car is parallel with the front wheels of the following car. This error is averted by counting three sets of wheels to be automatically count as two vehicles. However after counting the wheels we need to count the vehicles number. Wheel / 2 gives us this number as every vehicle has a set of 4 wheels and 2 of them are in parallel. This conversion is done in the 2nd arduino, which is the driver arduino of the GSM module. In this 2nd arduino's code there are some AT commands. The AT commands are just to setting the GSM module so that is can send the data to the server. It actually activate it's data connection, check the connection and send the data by hitting the server with GET method.

Chapter 04

Data Flow Structure and implementation

In this chapter, the concept of how the data, that is being collected by our vehicle counter, is being uploaded, downaloded, calculated upon and then finally uploaded again is presented. The data that is being sent by the counter are, number of vehicles crossing a particular point in five-minute interval, Longitude and Latitude of the counter and a time. Further into into the chapter we discuss how this set of data is being used to predict the traffic flow of a particular road.

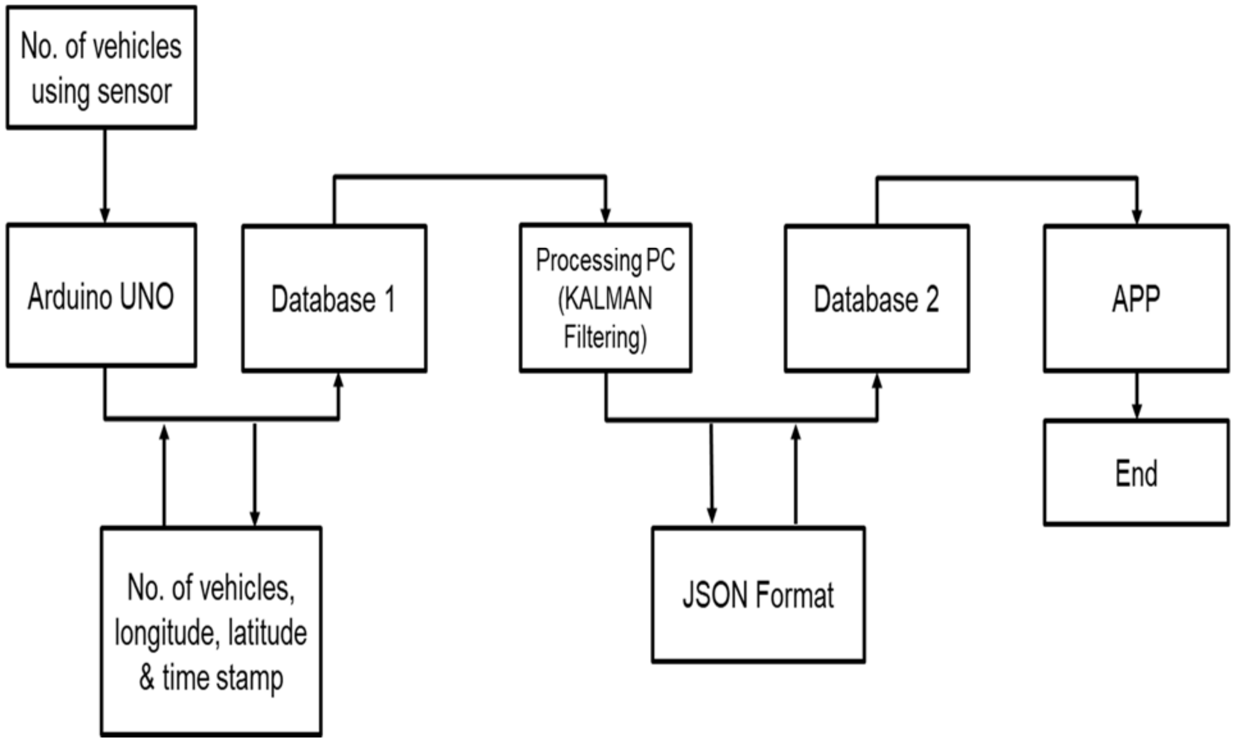

No. of vehicles using sensor
Arduino UNO
Database 1
Processing PC (KALMAN Filtering)
Database 2
APP
End
No. of vehicles, longitude, latitude & time stamp
JSON Format

Figure 5: The data flow chart.

4.1 Explanation

In this chapter, we will discuss how the data obtained from our field test will be processed. At first, our vehicle counter counts the number of cars crossing the road tube. This set of data will then be arranged periodically with 5 minute durations along with time stamps of when the data is being collected. Then it will be uploaded onto our server. From the server, we will download the data using Matlab's built in function [11] of *webread.* The Matlab software would then perform the Kalman Filtering technique to predict the number of cars that can pass through our point of data collection for the next week. These predicted values will be further used to calculate the traffic parameters for the next week in particular time periods. This final data of traffic parameters at different time will then be uploaded onto our server. This is achieved using the Matlab built in function *webwrite* [11]. After doing so, the predicted traffic flow rate will then be uploaded onto our custom made android application in the JSON (JavaScript Object Notation) format [12].

The servers mentioned here are the web domain we need to save or to store the data in the database. We purchased it from MucaHost. This gave us a .com website domain, database, FTP server, mail etc. We did not need all the facilities, we just needed the database and the .com domain. In the database we can stored the number of vehicles with respect to time and date. Firstly, we created a new database in our web domain's *phpmyadmin*. Then we created two tables for two specific work.

4.2 Database:

4.2.1 Database one:

We created out 1st table with the name of daily_data. This table contains ID, Day, Time, Route and ACC. This ID are the numbers of how many times the data came from the GSM or manually inserted. Then the day, which is the date of the data inserted. It's in YYYY-MM-DD format. After that the Time, here the time is not the actual time. This time is the desired time means the time when the last vehicle was counted. Suppose the last vehicle was counted in 12.05 pm which was total 168 vehicles. GSM module takes some time to process the number to upload in the database, network problem is also present in some places, so with all these problem data could be upload in 12.08 pm. However our desired time was 12.05pm, so we processed the time in such way so that the table shows the data came in 12.05 pm. This process was created in *php* format and uploaded in the *public_html*. It will be discussed later. In next column the header is route. Actually we wanted to use the GPS in the GSM module but because of high power consumption we did not use GPS and instead of it we used route number. Route number is for specifying a particular road, so that the number can make us understand that this route has now that number of vehicles. And the last column which is ACC. ACC stands for Actual Car Count. This is the main number, the actual number of vehicles that passes through the road for five minutes. In this table the main input is Route and ACC, so that we can understand that in this route tha number of vehicles just passed. The id, Day, Time is automatically inserted when Route and ACC come through the api code.

4.2.2 Database two:

This table is created with the name of kalman_data. This table contains id, Day, Time, Route, KPCC, Trafficflow, Density, Avgspacing, Headway, Error and Errorper. Like the previous (table one) here the table two's id, Day, Time and Route columns are same. KPCC is also similar to ACC but ACC was Actual Car Count and here KPCC is Kalman's Predicted Car Count. Normally the ACC was the integer number but the KPCC is decimal as it is filtered value. Filtered by Kalman filter with the help of Matlab. After that trafficflow, traffic flow the number of vehicles passed in a particular time. This one is also calculated by Matlab. Then vehicle density and average spacing. Vehicle density the the total number of vehicle that occupy a given length of a road and average spacing is the distance between successive vehicles in a traffic roadway. There are some more parameters such as headway, error and errorper. This headway means the the corresponding time between two successive vehicles as they pass a given point on the road way. Error is the difference between predicted vehicle number and counted vehicle number. These are also calculated in percentage. And this percentage is for errorper column. Here mentioned all the parameters are calculated in matlab and are uploaded in the database through an api code. This api code and the previous api code for table one are in public_html folder of our web domain's file manager folder.

4.3 Website:

In our thesis we had to buy a web domain to store the data on the database. We mentioned the uses of it earlier. The website contains pages like home, about, photos, hardware, Kalman filter, data, app and contact. In our first page, means in home page we have shown the importance of traffic prediction and the word flow of our whole process. In about page we mentioned why we are doing the things and what are the main parameters that are important for reducing traffic congestions. After that in our third page we showed some photos of counting the vehicles and where we measured the vehicle numbers. In hardware page we gave the operation principle of our counter and some photos of the components that we used in our vehicle counter. On fifth page our main part, Kalman filtering flow chart was shown. Here we have explained about the reasons of using Kalman filter. However after that we have a page which is really important. The data page. In this page you can see the data, the data that we had taken from the road and the calculated parameters. It was to show you the numbers of vehicle and other parameters if you just want to see and get the idea of a route. Later we gave our final android application in our APP page.

Chapter 05
Matlab Simulation and Calculation

In this chapter simulation result of the data set is presented in terms of analytical context. In the last chapter, we have seen how the data is being collected and made available for Matlab to utilize and output a predicted data set. But here we focus on the work of how the Matlab software takes a data reading, calculates the data using the Kalman Filter to output a predicted traffic flow count to make the whole project effective. We will further analyze them in order to check the performance and justifications of those calculated data.

5.1 Downloading Code

In our thesis so far we have uploaded data on our personal server (www.kalmanft.com) which has done by GSM module (SIM808). In our website we have many tabs on our homepage for instance Home, About, Photos, Hardware, Kalman ft, Data, App and Contact. When data is taken from the road it will straightly go to server Data page (http://www.kalmanft.com/data.php). Then we have to download all the data from server to matlab for kalman filtering which is our main focus of our thesis project. To download the data we have go to php my admin page (localhost/phpmyadmin) using a default browser. The computer must have wamp software to access this page otherwise we cannot go to the page. Then we have to give user and password otherwise anyoe can download data from any server which is prohibited. The user is ‘root’ and password is empty. After clicking go we can enter to the homepage of php my admin. On the left side we will find a database named ‘dailydata’. Under this database we will find a table named ‘daily_data’. Into that table we have our all data which has ID, Date, Time, Route and ACC (Actual Car Count) included there. After every five-minute data will be uploaded here by GSM Module. The data which are uploaded here is as a string. When we will download this data from here to matlab we have to do some process in the matlab. First, we have to open the matlab and then we have to apps tabs in matlab, then we have to go to database to connect with my php admin using ODBC connector. Then in the editor window we have to give connection with localhost and we have to download all the data from server to matlab. The problems we faced is while we were downloading the data it was downloading into cell matrix format but we needed only the ACC values in integer. To solve this problem, we have to download in the cell matrix format, then we have to convert into string and then we have convert it into integer. The matlab code for downloading is given below:

```
conn = database('dailydata','root','');
curs = exec(conn,'SELECT * FROM daily_data');
curs = fetch(curs);
a=curs.Data;
b = a(:,5);
c = b';
d=size(c);
e=d(2);
m=1;
for n = 1:e
    f = c{n};
    g = str2num(f);
    h(m)= g;
    m=m+1;
end
disp(h)
```

First matlab was downloading all the parameters then we extract only the ACC values which was in string mode. The ACC values was vertical aligned as for it was in the table in server. When we downloaded the all the information, which is ‘a’ in the code. From ‘a’ we can see ID, Date, Time, Route and ACC values are aligned in the table format. Then we extract only the ACC values which will be stored in ‘b’ in the matlab in vertical formation. However, we will transform value of ‘b’ from vertical to horizontal by transposing ‘b’ which is now ‘c’. The amount of data we have to convert from string into integer we need he number to run the for loop because we do not know the amount of how many days’ data will be there. So we have to run the ‘for’ loop from first value to how many values we have. So needed the number of values present in the server. In the code ‘d’ is the size of ‘c’ will tell us how many rows and how many columns are there. We are looking for columns because there is only one row but how many number of data is present there we can say it from column number. Thus, ‘e’ will tell us the number of columns ‘d’ has which is the number of data we have in the server. By this method we will know the number of data which is exactly ‘e’ in the code. Then we run the for loop where ‘n’ is the iteration number and it will continue the loop from 1 to ‘e’ which is the total data number. Then in the for loop we first break the cell matrix. The cell matrix was every single five minutes’ data itself which is 1*1 cell matrix. We break the cell matrix using ‘{}’ in the code. Now ‘f’ is the string value then we again have to convert it to integer which is ‘g’. Now finally we have the integer value, now we will save it to ‘h’ in array format. Then we do increment ‘m’ by 1. The number of data we have in the server, the same number of loops will be running and ACC data will be saved in the ‘h’ in array format in the matlab. We will need this

ACC values to calculate Kalman Predicted Car Count (KPCC) for the next week's same day and time also we have to calculate error, error percentage and another four parameters which are traffic flow, traffic density, Average spacing and Headway from the values of ACC and KPCC.

5.2 Kalman Filter Implementation

The Kalman Filter coding part is very important part in our thesis project. Because using this algorithm coding we can get the predicted next weeks' value and which is also very close to the real value we are getting from vehicle counter, which means our code is very much worthy and have a good trust issues. Using this code we can predict next weeks' same day value.The matlab Kalman code is given below:

```
X  = [164 180 170 165];
P  = 3;
R  = 4;

if(h(1)==X(1))
    h(1) = h(1)+1;
else
    h(1) = h(1);
end
u=1;
v=1;
for i=1:e
    K  = P/(P+R);
    r(v)  = X(i)+K*(h(u)-X(i));
    X(i)  = r(v);
    p  = (1-K)*P;
    P  = p;
    R  = abs(h(u)-X(i));
    u=u+1;
    v=v+1;
end
disp('KPCC values: '),disp(X)
```

The above code follows the following algorithm:

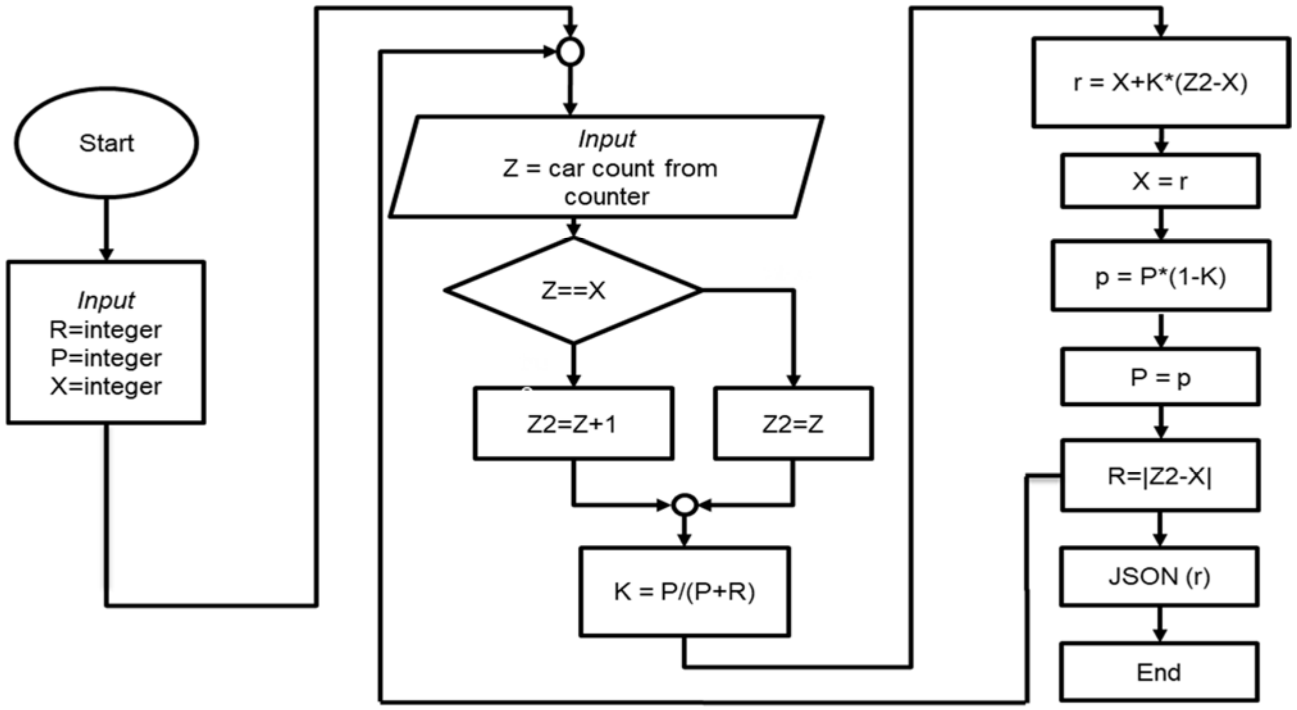


Figure 6: Matlab Kalman Filter Flow Chart

At first we assume previous weeks' vehicle number which is 'X' in the code, then we find error in estimation which is difference between first value of assumed value 'X' and first value of actual counting 'h' in the code and then we assume error in measurement 'R' which we assume very nearest value of error in estimation 'P'. then we first run the if loop which is for ensurement of first value of 'X' does not match with first value of 'h'. if two first value matched then we have to increase the the first value of 'h' by one. In the first line of if loop we have given a condition that if first value matched 'h(1)' will be increase by 1 and if not then in the else loop we remain it as the same value. Because if it is same the Kalman gain will be zero and next all calculations it will not give exact value. Then we assumed two variables 'u' and 'v'and assigned it with 1. It will help to store the value in array of ACC and KPCC because after run the simulation we want to see the KPCC values in array same as ACC was saved in array. Then we run the for loop. Here 'i' is the iteration in the loop. Then in the first line of loop we are measuring 'K' which is Kalman gain where we are diving error in estimation 'P' by addition of error in estimation 'P' and error in measurement 'R'. Next, we are calculating Kalman predicted value which is 'r' where weare subtracting KPCC values 'X' from ACC values 'h' and multiply with kalman gain 'K' and added with value of KPCC value. Now we are replacing the 'r' value with 'X'. Now 'X' is the new predicted Kalman value. Then we are calculating 'p' where we are doing (1-K) and multiply with old error in estimation which is ‘P’. this ‘p’ is new error in estimation and then this ‘p’ value is stored in ‘P’ which has replaced previous value of ‘P’. Then we are calculating ‘R’ which is the difference of absolute value of subtraction of KPCC from ACC value. Then we increase the array number ‘u’ and ‘v’ by 1. Moreover, the loop will continue until the final value of ACC come and calculated the KPCC value. After all KPCC values are calculated and stored in as array in ‘X’, it will come out from the loop and it will display KPCC value. This loop will be continued up to value of ‘e’ which is number of data we are getting from the server. That means each ACC values will be calculated here and KPCC will be the output.

5.3 Error Calculation

Here the error and error percentage code is given:

```
l=1;
o=1;
for j=1:e
    error(j)=abs(h(l)-X(o));
    errorper(j)=abs((h(l)-X(o))/h(l))*100;
    l=l+1;
    o=o+1;
end
disp('Error: '),disp(error)
disp('Error Percentage: '),disp(errorper)
```

from this code we can find the error and error percentage. First we will assume arbitrary array number variable ‘l’ and ‘o’ in the code. This array number will be used in ACC and KPCC value to calculate all the values of ACC and KPCC. Then we run a for loop where ‘j’ is the iteration number. In the first line ‘error’ is an array which will save every value coming from ACC and KPCC. In the code ‘error’ is the absolute value of subtraction of ACC and KPCC and then we find error percentage which is ‘errorper’ in the code. In this line we will subtract every ACC value from every KPCC value and then dividing it by that ACC value and then multiply with 100 to get the percentage value. Then we do increment of ‘l’ and ‘o’ because we need to increase the array number otherwise new value will replace old value or calculations can be wrong. Moreover, that is how we are continuing the for loop until every error and error percentage is counted from the use of ACC and KPCC values. Then it will be displayed in the matlab showing the value in array format with the error caption also same for the error percentage, it will show the value in array format along with the caption

5.4 Traffic Flow, Traffic Density, Average Spacing and Headway (ACC)

Now we need to find four parameters which are traffic flow, traffic density, average spacing and headway. These parameters will also be uploaded to server and traffic density will be shown in the android app for showing green and red line in the road to see the traffic is heavy or light. The matlab code to find traffic flow, traffic density, average spacing and headway from ACC values is given below:

```
t=1;
time=5;
length=4;
for s=1:e
    tfa(s)=h(t)/time;
    tda(s)=h(t)/length;
    asa(s)=length/h(t);
    hwa(s)=time/h(t);
    t=t+1;
end
disp('Traffic Flow for ACC: '),disp(tfa)
disp('Traffic Density for ACC: '),disp(tda)
disp('Average Spacing for ACC: '),disp(asa)
disp('Headway for ACC: '),disp(hwa)
```

here we are assigning variable ‘t’ with 1. It is an array number for ACC values to calculate four parameters. In the code ‘time’ is 5 which indicates five minutes. Which means after every minutes we want to see the traffic flow of the road. ‘length’ is 4 which is we measured traffic density of four meters. Now we run the for loop and here ‘s’ is the iteration number and loop will continue until it reaches total number of ACC values we have which is ‘e’ in the code. In the first line ‘’tfa’ which means Traffic Flow of ACC value is ACC value divided by time. If we divide number of cars by time, we get the flow. Same for here we are getting the traffic flow for ACC value. In the second lone we find ‘tda’ which means Traffic Density of ACC value. We can find it diving ACC values by length of the road. Next we are calculating ‘asa’ which is Average Spacing which is the inverse of Traffic Density. In the code we are diving length by ACC values which gives us the information of space between two vehicles. Next we are calculating ‘hwa’ which is inverse of Traffic Flow. Here we can find the time a vehicle takes to cross the road tube. For further calculation we can find the speed of the vehicle. In the code we are diving time by ACC value which is Headway for ACC values. Here ‘s’ stands for array number for ‘tfa’, ‘tda’, ‘asa’ and ‘hwa’ which starts with 1 and ends at ‘e’ which is the number of data we have in ACC. Then we do increment of t by 1. Then all four array ‘tfa’, ‘tda’, ‘asa’ and ‘hwa’ will be displayed in array in matlab.

5.5 Traffic Flow, Traffic Density, Average Spacing and Headway (KPCC)

Next we are going to find again four parameters which are traffic flow, traffic density, average spacing and headway for KPCC values. These parameters will also not be uploaded to server but we need this four parameters of KPCC to compare with ACC values that how close are these values. The matlab code to find traffic flow, traffic density, average spacing and headway from KPCC values is given below:

```
w=1;
for x=1:e
    tfk(x)=h(w)/time;
    tdk(x)=h(w)/length;
    ask(x)=length/h(w);
    hwk(x)=time/h(w);
    w=w+1;
end
disp('Traffic Flow for KPCC: '),disp(tfk)
disp('Traffic Density for KPCC: '),disp(tdk)
disp('Average Spacing for KPCC: '),disp(ask)
disp('Headway for KPCC: '),disp(hwk)
```

here we are assigning variable ‘w’ with 1. It is an array number for KPCC values to calculate four parameters. Same for previous code ‘time’ is 5 which indicates five minutes. Which means after every minutes we want to see the traffic flow of the road, ‘length’ is 4 which is we measured traffic density of four meters. Now we run the for loop and here ‘x’ is the iteration number and loop will continue until it reaches total number of KPCC values we have which is ‘e’ in the code. In the first line ‘’tfk’ which means Traffic Flow of KPCC value is KPCC value divided by time. If we divide number of cars by time, we get the flow. Same for here we are getting the traffic flow for KPCC value. In the second lone we find ‘tdk’ which means Traffic Density of KPCC value. We can find it diving KPCC values by length of the road. Next we are calculating ‘ask’ which is Average Spacing which is the inverse of Traffic Density. In the code we are diving length by KPCC values which gives us the information of space between two vehicles. Next we are calculating ‘hwk’ which is inverse of Traffic Flow. In the code we are diving time by KPCC value which is Headway for KPCC values. Here ‘x’ stands for array number for ‘tfk’, ‘tdk’, ‘ask’ and ‘hwk’ which starts with 1 and ends at ‘e’ which is the number of data we have in KPCC. Then we do increment of w by 1. Then all four array ‘tfk’, ‘tdk’, ‘ask’ and ‘hwk’ will be displayed in array in matlab with their respected caption.

5.6 Uploading Data

Now we have all ACC, KPCC value, error, error percentage along with four parameters which are Traffic Flow, Traffic Density, Average Spacing and Headway in array format. Now we have to just upload all the data for next procedure which is downloading data into Android app. To upload the data, we need the code of 'webwrite' which is:

```
response=webwrite(https://www.kalmanft.com/api/mathLabDataInput.php?Route='.$Route.'&Kp
cc='.$Kpcc.'&Trafficflow='.$Trafficflow.'&Density='.$Density.'&Avgspacing='.$Avgspacing
.'&Headway='.$Headway.'&Error='.$Error.'&Errorper='.$Errorper.',Route,1,Kpcc,176.5,Traf
ficflow,32.2,Density,40.25,Avgspacing,0.0248,Headway,0.031,Error,2,Errorper,3);
```

Here in the code 'url' means we have to write our api here. The api using which we want to upload data on the server. Then 'PostName1' is the title under which respective data will be uploaded. There are total eight sets of data we want to upload which are Route, KPCC, Traffic Flow, Traffic Density, Average Spacing, Headway, Error and Error percentage. These all data will be uploaded on the server. Then 'PostVavlue1' is the value of 'PostName1'. For instance, if we want upload KPCC data we will write KPCC in the 'PostName2' and we will write the value of KPCC in the 'PostValue1'. For the eight terms we have to write eight postname and eight postvalue serially. Then if we run the code all the data will be uploaded on the server and will be ready for next part which is downloading data using Android app to show the user the situation of the road which will be calculated using Traffic Density and it will show the green or red line on the android app to show heavy or light traffic.

Chapter 06

Data Analysis

To begin with this section of our thesis report we would demonstrate how the calculation of the collected data are being used to be displayed in tabular form and what their significance. This is very important if the data are readily available in tabular form for analysis which is what we have done in the data tables where we have placed all our data according to different categories and so it's easier to see and further analyze what they are about.

6.1) Comparing Actual Car Count and Kalman Predicted Car Count

The first table is where we have placed the actual car count (ACC) data along with the data that we got from the MATLAB through Kalman filter process and so that data is the Kalman Predicted car count (KPCC) data.

Time	Monday[1]		Monday[2]		Tuesday[1]		Tuesday[2]		Wednesday[1]		Wednesday[2]	
	ACC	KPCC	ACC	KPCC	ACC	KPCC	ACC	KPCC	ACC	KPCC	ACC	KPCC
8:30 – 8:35	161	158.23	159	158.23	115	158.23	179	158.23	171	158.23	155	158.23
8:35 – 8:40	163	161.71	152	161.71	145	161.71	145	161.71	155	161.71	176	161.71
8:40 – 8:45	183	180.73	152	180.73	134	180.73	152	180.73	202	180.73	158	180.73
8:45 – 8:50	195	192.73	158	192.73	141	192.73	148	192.73	105	192.73	180	192.73
8:50 – 8:55	163	163.33	164	163.33	215	163.33	183	163.33	164	163.33	187	163.33
8:55 – 9:00	135	137.27	122	137.27	190	137.27	181	137.27	187	137.27	179	137.27
9:00 – 9:05	184	182.22	146	182.22	176	182.22	193	182.22	199	182.22	220	182.22
9:05 – 9:10	185	186.29	163	186.29	182	186.29	176	186.29	132	186.29	154	186.29
9:10 – 9:15	200	190.24	159	190.24	186	190.24	163	190.24	174	190.24	193	190.24
9:15 – 9:20	175	165.24	135	165.24	128	165.24	149	165.24	141	165.24	140	165.24
9:20 – 9:25	178	173.24	148	173.24	161	173.24	173	173.24	182	173.24	211	173.24
9:25 – 9:30	185	188.76	156	188.76	154	188.76	169	188.76	165	188.76	240	188.76

Table 1 Number of cars passing through with respect to time

The table 01 which is showing the number of cars passing through with respect to time shows us how from time to time the data are collected after every 5 minutes and the most important thing is both the ACC and the KPCC values are place side by side according to date and time so that very easily we can find what could be the predicted amount of traffic from the KPCC of that day from the previous weeks data and thus know about the upcoming traffic condition of that particular day. Form the above table we see the data of the particular day of two consecutive weeks are placed and this data places the ACC and the KPCC side by side which helps us to find the traffic prediction. For example on the 1st week for the first time taste we obtained the KPCC value and this KPCC value can be used as an prediction for the next weeks same day at the same time. Say if anyone sees the table for Monday1 at 9.00am to 9.05am then it is possible for him to predict the traffic on the next Monday2 and this is what the table show us as we see the KPCC for Monday at 9.00am to 9.05 am is 182.22 whereas on the next Monday2 the value of the vehicles are found as ACC value: 146 and KPCC: 182.22 so the difference is quite reasonable if the adverse traffic environment are taken into consideration. This is how for each day of the week for every 5 minutes anyone can predict the traffic outcome from the analysis of these data.

6.2) Error and Error Percentage Comparison

The second table shows us the error measured between the KPCC and the ACC values which is calculated as:

$$\{(ACC-KPCC)/ACC\}*100\%$$

When we take a closer look we find errors are both positive and negative which indicates the prediction of the Kalman filter is always somewhere near the ACC either more (when error is negative) or less (when error is positive). This difference is caused by the uncertainty of the system as the environment is not always in our favor which is why our project is more likely to be taken into consideration for its practical and realistic outcome. This error is shown in the table below for both ACC and KPCC which are always close to each other.

Time	Monday[1]		Monday[2]		Tuesday[1]		Tuesday[2]		Wednesday[1]		Wednesday[2]	
	Error	Error Percentage	Error	Error Percentage	Error	Error Percentage	Error	Error Percentage	Error	Error Percentage	Error	Error Percentage
8:30 – 8:35	2.77	1.72	0.35	0.22	-11.76	-10.23	25.59	14.30	1.29	0.75	-6.37	-4.11
8:35 – 8:40	1.29	0.79	-4.16	-2.74	10.76	7.42	5.26	3.63	0.33	0.21	7.11	4.04
8:40 – 8:45	2.27	1.24	-13.06	-8.59	1.78	1.33	8.79	5.78	9.76	4.83	-16.70	-10.57
8:45 – 8:50	2.27	1.16	-15.79	-9.99	-2.77	-1.96	1.95	1.32	-26.75	-25.48	23.90	13.28
8:50 – 8:55	-0.33	-0.20	0.23	0.14	7.26	3.38	-11.97	-6.54	-3.27	-1.99	9.21	4.93
8:55 – 9:00	-2.27	-1.68	-6.94	-5.69	0.80	0.42	-3.28	-1.81	0.80	0.43	-2.88	-1.61
9:00 – 9:05	1.78	0.97	-16.10	-11.03	-1.78	-1.01	6.77	3.51	-4.76	-2.39	7.73	3.51
9:05 – 9:10	-1.29	-0.70	-9.98	-6.12	2.27	1.25	-1.69	-0.96	2.27	1.72	11.03	7.16
9:10 – 9:15	9.76	4.88	-15.24	-9.58	5.26	2.83	-8.48	-5.20	-12.25	-7.04	3.31	1.72
9:15 – 9:20	9.76	5.58	-14.75	-10.93	-3.76	-2.94	8.11	5.44	0.33	0.23	-0.22	-0.16
9:20 – 9:25	4.76	2.67	-12.02	-8.12	-3.76	-2.34	3.88	2.24	-1.29	-0.71	11.88	5.63
9:25 – 9:30	-3.76	-2.03	-15.42	-9.88	-0.33	-0.21	4.89	2.89	0.33	0.20	25.11	10.46

Table 2 Error and Error Percentage of ACC and KPCC

6.3) Traffic Parameters Comparison between ACC and KPCC

The table 03 provides us the information about parameters that are of huge significance when it comes to the calculation for the traffic flow, traffic density, average spacing and the headway. These parameters are of huge impact such as:

The traffic flow tells us the vehicle number for a particular time unit. For instance, in the given table of the data sheet we found the number of vehicles on the road on Tuesday were 179 according to our sensor value ACC and 158.23 (see Table 01 Number of cars passing through with respect to time) based on the kalman predicted car count value KPCC. For each case these data can be used as the value for the number of vehicles r and as mentioned the time range has been for 5 minutes which is 300 seconds. The formula for traffic flow gives the result to be 35.8 for ACC and for KPCC its 30.682 (see Table 3 Traffic Parameters table with Actual Car Count and Kalman Predicted Car Count). From those tables data we are having the idea how the traffic flow is varied with respect to time.

Secondly, the Traffic density which we define as the number of vehicles (r) occupying a unit length of roadway at an instant in time is illustrated in the following formula:

$$Vehicle\ Density = r/l$$

Where r is the total number of cars in a road (both from ACC and KPCC) occupying a given length of the roadway and l denotes the length of the roadway we have performed our field work. For Tuesday 1 from 8.30am to 8.35am the ACC is 115 and KPCC is 158.23 and for the road length in terms of lanes of the road is l =4. So we get in the table the density values for that particular time density =28.75 for ACC and in the same way the KPCC density is calculated as 31.69.

Moreover, the traffic headway meaning the corresponding time between two successive vehicles as they pass a given point on the roadway is calculated as:

$$Headway = 1/Traffic\ Flow$$

Using this formula, we calculated the ACC traffic Headway for Tuesday1 to be 0.0347 and as we know the headway puts much importance on the traffic condition improvement which is described clearly in the traffic flow parameters section in this report.

Calculating the parameter called Average Spacing which is the distance between successive vehicles in a traffic roadway and this is defined by the equation as:

$$Average\ Spacing = 1/Vehicle\ Density$$

Using this formula for Tuesday 1 we found the average spacing to be 0.03155 for ACC value and we have described how the average spacing term can be of great help in better traffic management and prediction.

In the table 3 all these four parameters are arranged and so it is stated as follows:

	Monday1								Monday2							
	ACC				KPCC				ACC				KPCC			
Time	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway
8:30 – 8:35	32.2	40.25	0.02484472	0.0310559	31.646	39.5575	0.02527966	0.03159957	31.8	39.75	0.025157233	0.0314465	31.73	39.6625	0.025212732	0.031516
8:35 – 8:40	32.6	40.75	0.024539877	0.0306748	32.342	40.4275	0.02473564	0.03091955	30.4	38	0.026315789	0.0328947	31.232	39.04	0.025614754	0.032018
8:40 – 8:45	36.6	45.75	0.021857923	0.0273224	36.146	45.1825	0.02213246	0.02766558	30.4	38	0.026315789	0.0328947	33.012	41.265	0.024233612	0.030292
8:45 – 8:50	39	48.75	0.020512821	0.025641	38.546	48.1825	0.02075442	0.02594303	31.6	39.5	0.025316456	0.0316456	34.758	43.4475	0.023016284	0.02877
8:50 – 8:55	32.6	40.75	0.024539877	0.0306748	32.666	40.8325	0.0244903	0.03061287	32.8	41	0.024390244	0.0304878	32.754	40.9425	0.024424498	0.030531
8:55 – 9:00	27	33.75	0.02962963	0.037037	27.454	34.3175	0.02913965	0.03642456	24.4	30.5	0.032786885	0.0409836	25.788	32.235	0.031022181	0.038778
9:00 – 9:05	36.8	46	0.02173913	0.0271739	36.444	45.555	0.02195149	0.02743936	29.2	36.5	0.02739726	0.0342466	32.42	40.525	0.024676126	0.030845
9:05 – 9:10	37	46.25	0.021621622	0.027027	37.258	46.5725	0.0214719	0.02683987	32.6	40.75	0.024539877	0.0306748	34.596	43.245	0.023124061	0.028905
9:10 – 9:15	40	50	0.02	0.025	38.048	47.56	0.02102607	0.02628259	31.8	39.75	0.025157233	0.0314465	34.848	43.56	0.022956841	0.028696
9:15 – 9:20	35	43.75	0.022857143	0.0285714	33.048	41.31	0.02420721	0.03025902	27	33.75	0.02962963	0.037037	29.95	37.4375	0.026711185	0.033389
9:20 – 9:25	35.6	44.5	0.02247191	0.0280899	34.648	43.31	0.02308936	0.02886169	29.6	37	0.027027027	0.0337838	32.004	40.005	0.024996875	0.031246
9:25 – 9:30	37	46.25	0.021621622	0.027027	37.752	47.19	0.02119093	0.02648866	31.2	39	0.025641026	0.0320513	34.284	42.855	0.0233345	0.029168

	Tuesday1								Tuesday2							
	ACC				KPCC				ACC				KPCC			
Time	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway
8:30 – 8:35	23	28.75	0.034782609	0.0434783	25.352	31.69	0.0315557	0.03944462	35.8	44.75	0.022346369	0.027933	30.682	38.3525	0.02607392	0.032592
8:35 – 8:40	29	36.25	0.027586207	0.0344828	26.848	33.56	0.02979738	0.03724672	29	36.25	0.027586207	0.0344828	27.948	34.935	0.028624589	0.035781
8:40 – 8:45	26.8	33.5	0.029850746	0.0373134	26.444	33.055	0.03025261	0.03781576	30.4	38	0.026315789	0.0328947	28.642	35.8025	0.02793101	0.034914
8:45 – 8:50	28.2	35.25	0.028368794	0.035461	28.754	35.9425	0.02782222	0.03477777	29.6	37	0.027027027	0.0337838	29.21	36.5125	0.027387881	0.034235
8:50 – 8:55	43	53.75	0.018604651	0.0232558	41.548	51.935	0.01925484	0.02406855	36.6	45.75	0.021857923	0.0273224	38.994	48.7425	0.020515977	0.025645
8:55 – 9:00	38	47.5	0.021052632	0.0263158	37.84	47.3	0.02114165	0.02642706	36.2	45.25	0.022099448	0.0276243	36.856	46.07	0.021706099	0.027133
9:00 – 9:05	35.2	44	0.022727273	0.0284091	35.556	44.445	0.02249972	0.02812465	38.6	48.25	0.020725389	0.0259067	37.246	46.5575	0.021478817	0.026849
9:05 – 9:10	36.4	45.5	0.021978022	0.0274725	35.946	44.9325	0.02225561	0.02781951	35.2	44	0.022727273	0.0284091	35.538	44.4225	0.022511115	0.028139
9:10 – 9:15	37.2	46.5	0.021505376	0.0268817	36.148	45.185	0.02213124	0.02766405	32.6	40.75	0.024539877	0.0306748	34.296	42.87	0.023326335	0.029158
9:15 – 9:20	25.6	32	0.03125	0.0390625	26.352	32.94	0.03035823	0.03794778	29.8	37.25	0.026845638	0.033557	28.178	35.2225	0.028390943	0.035489
9:20 – 9:25	32.2	40.25	0.02484472	0.0310559	32.952	41.19	0.02427774	0.03034717	34.6	43.25	0.023121387	0.0289017	33.824	42.28	0.023651845	0.029565
9:25 – 9:30	30.8	38.5	0.025974026	0.0324675	30.866	38.5825	0.02591849	0.03239811	33.8	42.25	0.023668639	0.0295858	32.822	41.0275	0.024373896	0.030467

	Wednesday1								Wednesday2							
	ACC				KPCC				ACC				KPCC			
Time	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway	Traffic Flow	Traffic Density	Avg. Spacing	Headway
8:30 – 8:35	34.2	42.75	0.023391813	0.0292398	33.942	42.4275	0.02356962	0.02946202	31	38.75	0.025806452	0.0322581	32.274	40.3425	0.024787755	0.030985
8:35 – 8:40	31	38.75	0.025806452	0.0322581	30.934	38.6675	0.02586151	0.03232689	35.2	44	0.022727273	0.0284091	33.778	42.2225	0.023684055	0.029605
8:40 – 8:45	40.4	50.5	0.01980198	0.0247525	38.448	48.06	0.02080732	0.02600916	31.6	39.5	0.025316456	0.0316456	34.94	43.675	0.022896394	0.02862
8:45 – 8:50	21	26.25	0.038095238	0.047619	26.35	32.9375	0.03036053	0.03795066	36	45	0.022222222	0.0277778	31.22	39.025	0.0256246	0.032031
8:50 – 8:55	32.8	41	0.024390244	0.0304878	33.454	41.8175	0.02391343	0.02989179	37.4	46.75	0.021390374	0.026738	35.558	44.4475	0.022498453	0.028123
8:55 – 9:00	37.4	46.75	0.021390374	0.026738	37.24	46.55	0.02148228	0.02685285	35.8	44.75	0.022346369	0.027933	36.376	45.47	0.021992523	0.027491
9:00 – 9:05	39.8	49.75	0.020100503	0.0251256	40.752	50.94	0.01963094	0.02453867	44	55	0.018181818	0.0227273	42.454	53.0675	0.018843925	0.023555
9:05 – 9:10	26.4	33	0.03030303	0.0378788	25.946	32.4325	0.03083327	0.03854159	30.8	38.5	0.025974026	0.0324675	28.594	35.7425	0.027977897	0.034972
9:10 – 9:15	34.8	43.5	0.022988506	0.0287356	37.25	46.5625	0.02147651	0.02684564	38.6	48.25	0.020725389	0.0259067	37.938	47.4225	0.021087037	0.026359
9:15 – 9:20	28.2	35.25	0.028368794	0.035461	28.134	35.1675	0.02843535	0.03554418	28	35	0.028571429	0.0357143	28.044	35.055	0.028526601	0.035658
9:20 – 9:25	36.4	45.5	0.021978022	0.0274725	36.658	45.8225	0.02182334	0.02727918	42.2	52.75	0.018957346	0.0236967	39.824	49.78	0.020088389	0.02511
9:25 – 9:30	33	41.25	0.024242424	0.030303	32.934	41.1675	0.02429101	0.03036376	48	60	0.016666667	0.0208333	42.978	53.7225	0.018614175	0.023268

Table 3 Traffic Parameters table with Actual Car Count and Kalman Predicted Car Count

6.4) Graphical representation of ACC and KPCC

There is no better way than to represent any sets of data through graphical presentation and that is why the following data are shown in the graph below:

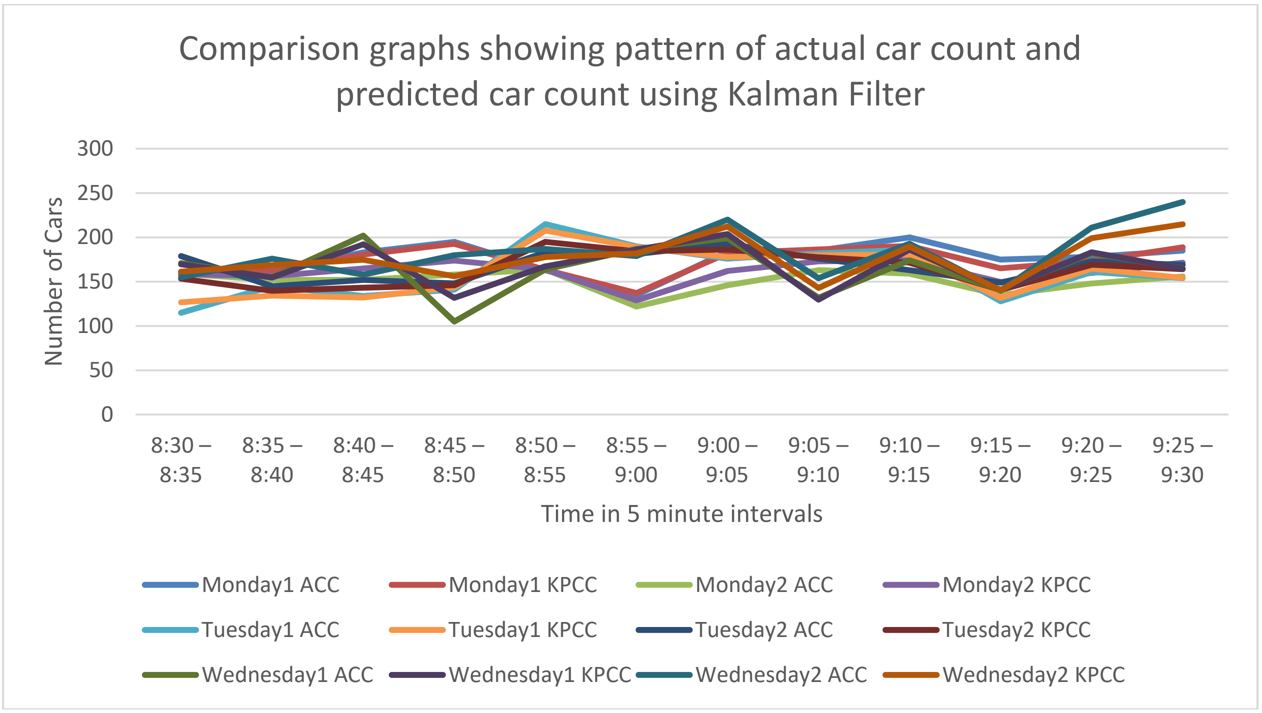


Figure 7: Comparison graphs showing pattern of ACC and KPCC

The figure above shows how the ACC and the KPCC values are following a pattern with respect to time. In the horizontal axis is the time whereas the vertical axis represents the number of vehicles within which both the reading ACC and KPCC varies every time. If we see in the graph for a particular section let's say the time between 8.50am to 855am and till 9.00am we find the Monday 1 KPCC value and the Monday 2 ACC values are following the almost same pattern(in the presence of certain system uncertainty) and also from the vertical axis we find the variation of the ACC and the KPCC to be very less and that's is why our kalman filter algorithm is best suited for the prediction of traffic number and condition when all other conditions are stable. To sum up this section we would put importance on how and what are the tables all about and how the values are being calculated. However this thing must be in consideration that with each ACC value we have placed the KPCC value side by side so that the kalman filter algorithm is clearly compared with the practical values and this is of great importance to evaluate the performance of the selected kalman filter system.

Chapter 07

Android Mobile Application

For developing the android application, we are going to need a set of application programming interfaces(API) such as the google maps API and a custom API for our server. After the calculated values as shown in Table. 2, are uploaded into our server, they will be accessed by our application using the Representational State Transfer API method or REST API method for short. Given our limited area of experimentation, we have drawn the route onto the google map using *Polyline* between two road junctions. This is done in order to highlight that our results are for a complete stretch of road where any loss or addition of cars is omitted. In the application, we have used the widget named radio button in order to help select the particular day for which the data is to be downloaded. We have also integrated a *Discrete SeekBar*. This *SeekBar* will help us to select the time whose data we would like to view. After the set of data is downloaded and displayed onto our *listview* upon clicking the data, the new *MapActivity* is launched. There the polyline colour will vary according to value of traffic density. The polyline colour, for example, will be red when the traffic density values are between 185-250 and blue when range is from 36-48.

7.1 Working Principles

The app starts with the first *Activity* which is the first page of the application. Here, the user selects their desired day by selecting the *radio buttons.* After selecting the day, they can select the time using the *Seekbar.* After pressing the select day button, the date and time variables are selected and the equivalent data is downloaded from the server using the REST API method. The data is fetched and parsed onto the application via the *volley GET* method of the volley library. This library a google recognized library which downloads the contents of a database in JSON format using the rest api architecture. The data is shown in a *listview* after the data is parsed and pushed into the array adapter, which is set into the list view. The *arraylist* is initialized inside the adapter which in turn is set and shown in the *listview*. The data is fetched using the GET request. JSON array request is used by the library to request the URL link to share the contents. The way the radio buttons and the *seekbar* filter out the data is quite simple. The main conditional loop starts by the selection of the radio button. If the radio button selected, for example for Sunday, then the fetching of data from the API *URL* is from the Sunday URL. Then while the data is being fetched, another *if* condition determines which particular time dependent data is to be selected depending on the position of that data in the JSON array format. The downloaded data is then displayed onto list view just below the radio buttons. The data shown are *Time, KPCC and Vehcle Denstiy.* Using the *listview intent*, this enables the user to tap on the displayed data which will take them onto the second activity which has the Google Maps API installed. Here, we will be able to see the stretch of road for which we have collected the field data. A *polyline* is drawn using the *polyoptions* method. The data which is passed from the main activity onto the maps activity is the Vehicle Density for that particular time and day. Saving this data onto a variable enables us to set a condition for which the colour of the polyline will vary. We have used a condition of vehicle density between 36-48 cars per lane-mile to denote that the road is not clogged with traffic jam, while a density of 100+ the polyline colour is changed to red which denotes that there is traffic jam present.

7.2 *Key Features Used*

7.2.1) Radiobuttons

```
public void onClickListenerRadioButton() {
    radio_g = (RadioGroup)findViewById(R.id.days);
    button_sbm = (Button)findViewById(R.id.day_sbm);
button_sbm.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    int selected_id = radio_g.getCheckedRadioButtonId();
                    button_sbm = (RadioButton)findViewById(selected_id);
                    Toast.makeText(MainActivity.this,
                            button_sbm.getText().toString(),Toast.LENGTH_SHORT
).show();
                    getDataFromServer();
                }
            }
    );
}
```

In the above code segment, we are able to see that upon clicking the radio buttons and clicking the button designated for saving the selection, the Id of the radio buttons are saved into the button named *button_sbm*. This enables us to select multiple days of the day one by one in order to fetch the data more accurately.

7.2.2) Seekbar

```
public void seebbarr( ){
    seek_bar = (SeekBar)findViewById(R.id.seekBar);
    text_view =(TextView)findViewById(R.id.textView);
    text_view.setText("Covered : " + seek_bar.getProgress() + " / "
+seek_bar.getMax());
progress_value = progress;
```

Here, the *seekbar* progress is saved onto the variable named *progress_value*. The *seekbar* progress is the amount of length covered by the *seekbar*. This *progress_value* is used to save the progress of the *seekbar* in order to select the time.

7.2.3) Volley

```
private void getDataFromServer() {
    Log.d("inside getData","before condition");
    if (button_sbm.getText().equals("Sun")) {
        JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
                GET_URL1, new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray jsonArray) {
                int length = jsonArray.length();

                if ((progress_value >= 1) && (progress_value <= 100)) {
                    try {
                        String time =
jsonArray.getJSONObject(progress_value).getString("user_id");
                        String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                        String density =
jsonArray.getJSONObject(progress_value).getString("Density");

                        arrayList.add(time + "\n" + kpcc + "\n" + density);
                        adapter.notifyDataSetChanged();
                    }
                    catch (JSONException e) {
                        Log.e("Error", e.toString());
                    }
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });

        requestQueue.add(jsonArrayRequest);
    }
```

Here we can see that, the *button_sbm* is checked according to the condition whether the radio button saved into is “sun” or not. If yes, then the code enters into this loop enabling the app to download the values for Sunday only. Furthermore, the *seekbar* progress is set as a condition whether there are any values selected or not. If so, then that progress value is used to retrieve the data from the Sunday table according to the position of the JSON data. For example, the time for

8:30 is saved in the first slot of the JSON data. Hence, when the *progress_value* is 1; it selects the entire row for the 8:30 AM time. Which includes the time, KPCC value and the Vehicle Density. This is shown in the *listview* of the app. The method used is the *GET* method of the Volley library. The URL used is *URL_1* which is the designated URL for hosting the Sunday values of time, KPCC and Vehicle Density.

7.2.4) Listview intent

```
private AdapterView.OnItemClickListener onListClick=new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent,
                            View view, int position,
                            long id)
    {
        //create our intent, as per usual
        Intent i=new Intent(MainActivity.this, MapsActivity.class);

        i.putExtra(ID_EXTRA, String.valueOf(id));
        startActivity(i);

    }
};
```

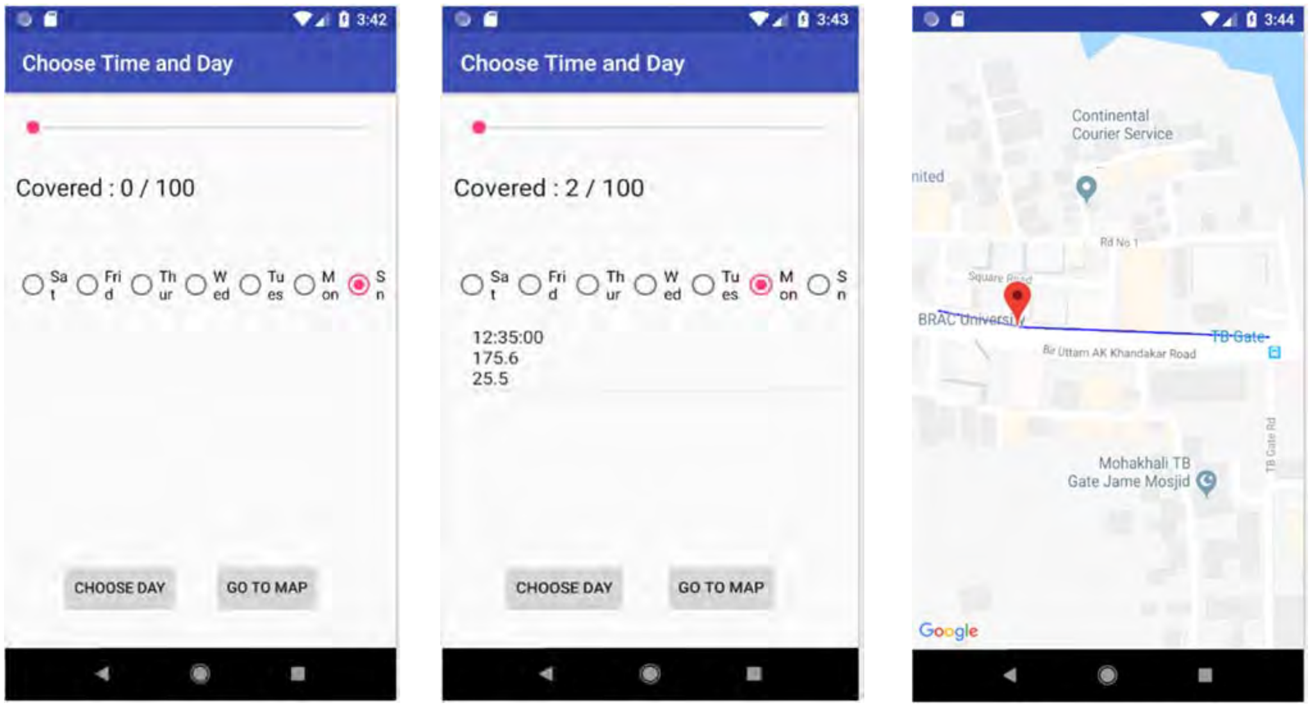
Here, the individual items of the *listview* is initiated using a special feature of ID_EXTRA. This put the value of the selected list view item into the intent which in turn opens the second activity along with a few values of that item.

7.2.5) Getting intent from previous activity

```
passedVar=getIntent().getStringExtra(MainActivity.ID_EXTRA);
```

the *getIntent* enables the second activity to receive and save the values coming in from the first activity onto the second activity into the variable named *passedVar*. This variable is then used to check the condition whether the density is within 36-48 to highlight the polyline blue, 100+ to highlight the polyline red.

7.3 **Final Result**



Covered : 0 / 100

Covered : 2 / 100

12:35:00
175.6
25.5

Figure 8: Android Application User Interface

The first screenshot is that of the first activity of the application. Here we see the *seekbar* along with the radio buttons and the choose day button. The white space between the radio buttons and bottom two buttons is the designated space where the *listview* is set. The downloaded data is displayed into this place.

The above screenshot displays that the time, day is selected and the choose day button is pressed. This enables the app to download the time and day specific data. The downloaded data is displayed into the list view. From the above example we can see that the selected day and time is Monday, 12:35 PM. This time has a KPCC value of 175.6 with a vehicle density of 25.5 which suggests that there is no traffic jam

After pressing the displayed data, the second intent with the Google Maps API is displayed. Since the traffic Density is lower than the density range set for traffic congestion, the polyline has a colour of blue suggesting the user that there is no traffic congestion.

Chapter 08

Conclusions and Future Work

FUTURE WORKS

In the future, we hope to modify the Kalman Filter by introducing other variables which would further improve the algorithm in order to predict the car count with better accuracy. Added to the improvement of the Kalman algorithm, we plan to improve the efficiency of the counter by using a better microcontroller than the generic AtMega microcontrollers used by the Arduino. The usage of camera based road counters too are in consideration in order to increase efficiency. In scenarios where the traffic is completely stagnant, in that case we will have a very low car count. In that case, we are planning on implementing an off-peak, on-peak system where the data from the previous weeks are directly implemented. This is still under consideration and is subject to further practical experimentation. A proper scale for determining the traffic condition too is required for easier understanding.

CONCLUSIONS

Starting with our motivation to help the human civilization get rid of traffic congestion, we have made a start in what we believe to be the future of traffic control. Collecting practical data, implementing that data onto the Kalman Filter Algorithm is just the beginning of our project. With the completion of our mobile application, it will truly aid daily commuters to make a reasonable decision about their next day routes. We believe, this type of future planning is required in order to avoid traffic gridlocks. Furthermore, from our calculated data, we are able to notice a systematic pattern being developed from Figure 5. This signifies that traffic flows are time dependant. The graphical representation of our data it is seen that the kalman filter has been able to predict the number of car count to a high degree of efficiency which we believe can be further enhanced by the use of other practical variables. If further proper research is done regarding this topic, we may very well have a scope to better the lives of billions across the globe.

References

[1] "Study: Dhaka traffic wastes 5 million work hours, costs Tk37,000 crore," The Dhaka Tribune (2018). [Online]. Available: https://www.dhakatribune.com/bangladesh/dhaka/2018/05/20/study-dhaka-traffic-wastes-5-million-work-hours-costs-tk37-000-crore

[2] Luis Ramon Leon Ojeda, Alain Y. Kibangou and Carlos Canudas de Wit, "Adaptive Kalman Filtering for Multi-Step ahead Traffic Flow Prediction," 2013 American Control Conference (ACC) Washington, DC, USA, June 17-19, 2013

[3] Zeeshan Hameed Mir and Fethi Filalil, "An Adaptive Kalman Filter based Traffic Prediction Algorithm for Urban Road Network", 2016 12th International Conference on Innovations in Information Technology (IIT)

[4] Ahmad Faisal Abidin, Mario Kolberg and Amir Hussain, "Improved Traffic Prediction Accuracy in Public Transport Using Trusted Information in Social Networks," Department of Computing Science and Mathematics, University of Stirling, UK {faa,mko,ahu}@cs.stir.ac.uk.

[5] Hans van Lint and Tamara Djukic, "Applications of Kalman Filtering in Traffic Management and Control," 2012 INFORMS isbn 978-0-9843378-3-5.

[6] Xiaotian Sun, Laura Muñoz, and Roberto Horowitz, "Mixture Kalman Filter Based Highway Congestion Mode and Vehicle Density Estimator and its Application," Proceeding of the 2004 American Control Conference Boston, Massachusetts June 30 - July 2, 2004.

[7] Selvaraj Vasantha Kumar, "Traffic Flow Prediction using Kalman Filtering Technique," 10th International Scientific Conference Transbaltica 2017: Transportation Science and Technology.

[8] https://www.youtube.com/watch?v=x1HM3IvExJE [Accessed: 6th June, 2018]

[9] https://github.com/neuroprod/Road-Tube-Traffic-Counter/blob/master/TrafficCounter.ino [Accessed: 6th June, 2018]

[10] https://www.arduino.cc/reference/en/language/functions/time/millis/ [Accessed: 6th June, 2018]

[11] https://www.mathworks.com/help/matlab/ref/webread.html [Accessed: 6th June, 2018]

[12] https://en.wikipedia.org/wiki/JSON [Accessed: 6th June, 2018]

[13] "Traffic Flow" https://en.wikipedia.org/wiki/Traffic_flow#Speed [Accessed: 6th June, 2018]

[14] https://lost-contact.mit.edu/afs/eos.ncsu.edu/info/ce400_info/www2/flow1.html [Accessed: 6th June, 2018]

[15] https://en.wikipedia.org/wiki/Traffic_congestion [Accessed: 6th June, 2018]

[16] https://www.mathworks.com/help/matlab/ref/webwrite.html [Accessed: 6th June, 2018]

[17] https://www.mathworks.com/help/database/ug/database.odbc.connection.exec.html [Accessed: 6th June, 2018]

[18] https://www.youtube.com/watch?v=X9cC0o9viTo [Accessed: 6th June, 2018]

[19] https://en.wikipedia.org/wiki/Kalman_filter [Accessed: 6th June, 2018]

[20] https://www.youtube.com/watch?v=tk3OJjKTDnQ [Accessed: 6th June, 2018]

[21] https://www.youtube.com/watch?v=fpRb1sjFz_M [Accessed: 6th June, 2018]

[22] https://www.youtube.com/watch?v=X9cC0o9viTo&amp=&t=8s [Accessed: 6th June, 2018]

[23] https://www.cs.cornell.edu/courses/cs4758/2012sp/materials/MI63slides.pdf [Accessed: 6th June, 2018]

APPENDIX

a. The following code is for the 1st arduino to count the number of wheels, see [9]

```
#include <Wire.h>

int ledPin = 12;
int sensorPin = A0;

int sensorValue = 0;
float averageValue = 0;
int count=0;

char t[10];
int del = 160;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
  Wire.begin();
}

void loop(){
      carc();
}

void carc() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  if(millis()<2000)
  {
      averageValue +=((float)sensorValue-averageValue)/10.f;

      digitalWrite(ledPin,HIGH);
  }else
  {
    if(sensorValue>averageValue +4){
      count=1;
      Serial.println(count);
      dtostrf(count, 1 , 0 , t);
      Wire.beginTransmission(9); // transmit to device #9
      Wire.write(t);              // sends x
      Wire.endTransmission();

      averageValue =sensorValue+10;

      digitalWrite(ledPin,HIGH);
      delay(del);
    }
    else
    {
        digitalWrite(ledPin,LOW);
        averageValue +=((float)sensorValue-averageValue)/100.f;
    }
  }
}
```

b. The following code is for the 2nd arduino to count the vehicles and uploading data to database: see [8]

```
#include <Wire.h>

#include <LiquidCrystal.h>
char car[20];
int minit;
const int rs = 2, en = 3, d4 = 4, d5 = 5, d6 = 6, d7 = 7;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

int8_t answer;
int counter;
String t;
String d;
char aux_str[200];
char aux;

int ledPin = 12;
int count= 0;
int car_count =0;
float car_countf =0;

void setup() {

  Serial.begin(9600);
  Wire.begin(9);
  Wire.onReceive(receiveEvent);
  setNetworkGSM();
```

```
 pinMode(ledPin, OUTPUT);
 pinMode(8, OUTPUT);
 digitalWrite(8, HIGH);
 lcd.begin(16, 2);

}

void loop() {
 // read the value from the sensor:
 delay(1000);
 digitalWrite(ledPin,LOW);

 minit++;
 if(minit>300){
  Send_data();
  minit=0;
  count= 0;
  car_count =0;
  car_countf =0;
 }
}

void Send_data(){
    dtostrf(car_count,1,2,car);
    answer = sendATcommand("AT+HTTPINIT", "OK", 20000);

    if (answer == 1)
    {
       answer = sendATcommand("AT+HTTPPARA=\"CID\",1", "OK", 5000);
       if (answer == 1)
       {
```

```
sprintf(aux_str,"AT+HTTPPARA=\"URL\",\"https://www.kalmanft.com/api/sendDataToDat
abase.php?route=1&acc=%s",car);

          Serial.print(aux_str);
          answer = sendATcommand("\"", "OK", 6000);
          if (answer == 1)
          {
            // Starts GET action
            answer = sendATcommand("AT+HTTPACTION=0", "+HTTPACTION:0,200",
10000);
            if (answer == 0)
            {
              Serial.println(F("Done!"));
            }
            else
            {
              Serial.println(F("Error getting url"));
            }
          }
          else
          {
            Serial.println(F("Error setting the url"));
          }
        }
        else
        {
          Serial.println(F("Error setting the CID"));
        }
      }
      else
```

```
        {
           Serial.println(F("Error initializating"));
        }
        sendATcommand("AT+HTTPTERM", "OK", 5000);
        minit=0;
}

int8_t sendATcommand(char* ATcommand, char* expected_answer, unsigned int timeout)
{
  uint8_t x=0,  answer=0;
  char response[100];
  unsigned long previous;

  memset(response, '\0', 100);    // Initialize the string

  delay(100);

  while( Serial.available() > 0) Serial.read();    // Clean the input buffer

  Serial.println(ATcommand);    // Send the AT command

    x = 0;
  previous = millis();

  // this loop waits for the answer
  do{

    if(Serial.available() != 0){
      response[x] = Serial.read();
      x++;
      // check if the desired answer is in the response of the module
      if (strstr(response, expected_answer) != NULL)
      {
        answer = 1;
      }
    }
    // Waits for the answer with time out
  }
  while((answer == 0) && ((millis() - previous) < timeout));
  return answer;
}

void setNetworkGSM()
{
  //delay(2000);
  while(1)
  {
    delay(15000);
    while (sendATcommand("AT+CREG?", "+CREG: 0,1", 2000) == 0);
    sendATcommand("AT+SAPBR=3,1,\"Contype\",\"GPRS\"", "OK", 2000);
    sendATcommand("AT+SAPBR=3,1,\"APN\",\"WAP\"", "OK", 2000);   //internet was
default
    sendATcommand("AT+SAPBR=3,1,\"USER\",\"\"", "OK", 2000);
    sendATcommand("AT+SAPBR=3,1,\"PWD\",\"\"", "OK", 2000);
    //int sapbrTest = 0;
    if(sendATcommand("AT+SAPBR=1,1", "OK", 20000) != 0)
    {
      break;
    }
  }
}
```

```
void receiveEvent(int bytes) {
 int i = 0;
 while (Wire.available()) {
   t[i] = Wire.read(); // every character that arrives it put in order in the empty array "t"
   i=i+1;
   digitalWrite(ledPin,HIGH);
 }
 count++;
 Serial.print("wheels:");
 Serial.println(count);
    car_countf = (float)count/2.f;
    car_count = count/2;
    float last = (car_countf-car_count);

    if(last>0){
      car_count+=1;
      Serial.print("vehicle:");
      Serial.println( car_count);
      lcd.setCursor(0,0);
      lcd.print("Vehicles:");
      lcd.print(car_count);
    }
}
```

c. API code of uploading data through GSM module:

```
https://www.kalmanft.com/api/sendDataToDatabase.php?route='.$route.'&acc='.$acc;
```

d. API code of uploading data through Matlab Simulink:

```
https://www.kalmanft.com/api/mathLabDataInput.php?Route='.$Route.'&Kpcc='.$Kpcc.'&T
rafficflow='.$Trafficflow.'&Density='.$Density.'&Avgspacing='.$Avgspacing.'&Headway='.
$Headway.'&Error='.$Error.'&Errorper='.$Errorper;
```

e. API for downloading data into Android Application

https://www.kalmanft.com/api/getDataForAndroid.php

f. Matlab Downloading Code

```
conn = database('dailydata','root','');
curs = exec(conn,'SELECT * FROM daily_data');
curs = fetch(curs);
a=curs.Data;
b = a(:,5);
c = b';
d=size(c);
e=d(2);
m=1;
for n = 1:e
   f = c{n};
   g = str2num(f);
   h(m)= g;
   m=m+1;
end
     disp(h)
```

g. Matlab Kalman Filter Code

```
X  = [164 180 170 165];
P  = 3;
R  = 4;

if(h(1)==X(1))
   h(1) = h(1)+1;
else
   h(1) = h(1);
end
```

```
u=1;
v=1;
for i=1:e
   K  = P/(P+R);
   r(v)  = X(i)+K*(h(u)-X(i));
   X(i)  = r(v);
   p  = (1-K)*P;
   P  = p;
   R  = abs(h(u)-X(i));
   u=u+1;
   v=v+1;
end
disp('KPCC values: '),disp(X)
```

h. Matlab Error Code

```
l=1;
o=1;
for j=1:e
   error(j)=abs(h(l)-X(o));
   errorper(j)=abs((h(l)-X(o))/h(l))*100;
   l=l+1;
   o=o+1;
end
disp('Error: '),disp(error)
disp('Error Percentage: '),disp(errorper)
```

i. Matlab Traffic Flow, Traffic Density, Average Spacing and Headway (ACC)

```
t=1;
time=5;
length=4;
```

```
for s=1:e
   tfa(s)=h(t)/time;
   tda(s)=h(t)/length;
   asa(s)=length/h(t);
   hwa(s)=time/h(t);
   t=t+1;
end
disp('Traffic Flow for ACC: '),disp(tfa)
disp('Traffic Density for ACC: '),disp(tda)
disp('Average Spacing for ACC: '),disp(asa)
disp('Headway for ACC: '),disp(hwa)
```

j. Matlab Traffic Flow, Traffic Density, Average Spacing and Headway (KPCC)

```
w=1;
for x=1:e
   tfk(x)=h(w)/time;
   tdk(x)=h(w)/length;
   ask(x)=length/h(w);
   hwk(x)=time/h(w);
   w=w+1;
end
disp('Traffic Flow for KPCC: '),disp(tfk)
disp('Traffic Density for KPCC: '),disp(tdk)
disp('Average Spacing for KPCC: '),disp(ask)
disp('Headway for KPCC: '),disp(hwk)
```

k. Matlab Uploading Codes

```
response=webwrite(https://www.kalmanft.com/api/mathLabDataInput.php?Route='.$Route.'
&Kpcc='.$Kpcc.'&Trafficflow='.$Trafficflow.'&Density='.$Density.'&Avgspacing='.$Avgsp
acing.'&Headway='.$Headway.'&Error='.$Error.'&Errorper='.$Errorper.',Route,1,Kpcc,176.5
,Trafficflow,32.2,Density,40.25,Avgspacing,0.0248,Headway,0.031,Error,2,Errorper,3);
```

l. Android Main Activity

```
package lict.nsu.volleyexample;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonArrayRequest;
import com.android.volley.toolbox.Volley;

import org.json.JSONArray;
import org.json.JSONException;
import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

    ListView listView;
    ArrayAdapter<String> adapter;
    ArrayList<String> arrayList;
    static final String GET_URL1 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL2 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL3 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL4 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL5 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL6 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    static final String GET_URL7 = "https://www.kalmanft.com/api/getDataForAndroid.php";
    RequestQueue requestQueue;
    private static SeekBar seek_bar;
    private static TextView text_view;
    int progress_value;
    private static RadioGroup radio_g;
    private static RadioButton radio_day;//, radio_day2, radio_day3, radio_day4, radio_day5,
radio_day6, radio_day7;
    private static Button button_sbm;
    /*RadioButton r1;
    RadioButton r2;
    RadioButton r3;
    RadioButton r4;
    RadioButton r5;
    RadioButton r6;
    RadioButton r7;*/
    public  static final String ID_EXTRA = "lict.nsu.volleyexample._ID";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        seebbarr( );
        onClickListenerRadioButton();

        listView = findViewById(R.id.listView);
        arrayList = new ArrayList<>();
        adapter = new ArrayAdapter<String>(MainActivity.this,
android.R.layout.simple_list_item_1,
                arrayList);
        listView.setAdapter(adapter);
        listView.setOnItemClickListener(onListClick);
        //Log.d(String, "onCreate: SThs");
        requestQueue = Volley.newRequestQueue(MainActivity.this);
        getDataFromServer();




        }



        //radio_g = (RadioGroup) findViewById(R.id.days);
        /*radio_g.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(RadioGroup radioGroup, int i) {

                RadioButton r1 = (RadioButton) findViewById(R.id.radioButton1);
                RadioButton r2 = (RadioButton) findViewById(R.id.radioButton2);
                RadioButton r3 = (RadioButton) findViewById(R.id.radioButton3);
                RadioButton r4 = (RadioButton) findViewById(R.id.radioButton4);
                RadioButton r5 = (RadioButton) findViewById(R.id.radioButton5);
                RadioButton r6 = (RadioButton) findViewById(R.id.radioButton6);
                RadioButton r7 = (RadioButton) findViewById(R.id.radioButton7);

            }


        });*/


    private AdapterView.OnItemClickListener onListClick=new
AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent,
                            View view, int position,
                            long id)
        {
            //create our intent, as per usual
            Intent i=new Intent(MainActivity.this, MapsActivity.class);

            i.putExtra(ID_EXTRA, String.valueOf(id));
            startActivity(i);

        }
```

```
};



public void onClickListenerRadioButton() {
    radio_g = (RadioGroup)findViewById(R.id.days);
    button_sbm = (Button)findViewById(R.id.day_sbm);
    //radio_day =
(RadioButton)findViewById(R.id.radioButton1);//findViewById(R.id.radioButton2);findVie
wById(R.id.radioButton3);findViewById(R.id.radioButton4);findViewById(R.id.radioButto
n5);findViewById(R.id.radioButton6);findViewById(R.id.radioButton7);
    /*radio_day2 = (RadioButton)findViewById(R.id.radioButton2);
    radio_day3 = (RadioButton)findViewById(R.id.radioButton3);
    radio_day4 = (RadioButton)findViewById(R.id.radioButton4);
    radio_day5 = (RadioButton)findViewById(R.id.radioButton5);
    radio_day6 = (RadioButton)findViewById(R.id.radioButton6);
    radio_day7 = (RadioButton)findViewById(R.id.radioButton7);*/

    button_sbm.setOnClickListener(
            new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    int selected_id = radio_g.getCheckedRadioButtonId();
                    button_sbm = (RadioButton)findViewById(selected_id);
                    Toast.makeText(MainActivity.this,
                            button_sbm.getText().toString(),Toast.LENGTH_SHORT ).show();
                    getDataFromServer();
                }
            }
    );
}
```

```
   /*public void onRadioButtonClicked(View view) {
      // Is the button now checked?
      boolean checked = ((RadioButton) view).isChecked();
   }*/

    private void getDataFromServer() {
       Log.d("inside getData","before condition");
       if (button_sbm.getText().equals("Sun")) {
          JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
                GET_URL1, new Response.Listener<JSONArray>() {
             @Override
             public void onResponse(JSONArray jsonArray) {
                int length = jsonArray.length();

                if ((progress_value >= 1) && (progress_value <= 100)) {
                   try {
                      String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                      String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                      String density =
jsonArray.getJSONObject(progress_value).getString("Density");


                      arrayList.add(time + "\n" + kpcc + "\n" + density);
                      adapter.notifyDataSetChanged();
                   }
                   catch (JSONException e) {
                      Log.e("Error", e.toString());
                   }
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });

        requestQueue.add(jsonArrayRequest);
    }

    if (button_sbm.getText().equals("Mon")) {
        JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
                GET_URL2, new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray jsonArray) {
                int length = jsonArray.length();

                if ((progress_value >= 1) && (progress_value <= 100)) {
                    try {
                        String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                        String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                        String density =
jsonArray.getJSONObject(progress_value).getString("Density");

                        arrayList.add(time + "\n" + kpcc + "\n" + density);
                        adapter.notifyDataSetChanged();
                    } catch (JSONException e) {
```

```
                    Log.e("Error", e.toString());
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    requestQueue.add(jsonArrayRequest);
}
if (button_sbm.getText().equals("Tues")) {
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
            GET_URL3, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray jsonArray) {
            int length = jsonArray.length();

            if ((progress_value >= 1) && (progress_value <= 100)) {
                try {
                    String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                    String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                    String density =
jsonArray.getJSONObject(progress_value).getString("Density");

                    arrayList.add(time + "\n" + kpcc + "\n" + density);
                    adapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    Log.e("Error", e.toString());
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    requestQueue.add(jsonArrayRequest);
}
if (button_sbm.getText().equals("Wed")) {
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
            GET_URL4, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray jsonArray) {
            int length = jsonArray.length();

            if ((progress_value >= 1) && (progress_value <= 100)) {
                try {
                    String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                    String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                    String density =
jsonArray.getJSONObject(progress_value).getString("Density");

                    arrayList.add(time + "\n" + kpcc + "\n" + density);
                    adapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    Log.e("Error", e.toString());
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    requestQueue.add(jsonArrayRequest);
}
if (button_sbm.getText().equals("Thur")) {
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
            GET_URL5, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray jsonArray) {
            int length = jsonArray.length();

            if ((progress_value >= 1) && (progress_value <= 100)) {
                try {
                    String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                    String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                    String density =
jsonArray.getJSONObject(progress_value).getString("Density");
```

```
                    arrayList.add(time + "\n" + kpcc + "\n" + density);
                    adapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    Log.e("Error", e.toString());
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    requestQueue.add(jsonArrayRequest);
}
if (button_sbm.getText().equals("Frid")) {
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
            GET_URL6, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray jsonArray) {
            int length = jsonArray.length();

            if ((progress_value >= 1) && (progress_value <= 100)) {
                try {
                    String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                    String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
                    String density =
jsonArray.getJSONObject(progress_value).getString("Density");
```

```
                        arrayList.add(time + "\n" + kpcc + "\n" + density);
                        adapter.notifyDataSetChanged();
                    } catch (JSONException e) {
                        Log.e("Error", e.toString());
                    }
                }
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {

            }
        });

        requestQueue.add(jsonArrayRequest);
    }
    if (button_sbm.getText().equals("Sat")) {
        JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(Request.Method.GET,
                GET_URL7, new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray jsonArray) {
                int length = jsonArray.length();

                if ((progress_value >= 1) && (progress_value <= 100)) {
                    try {
                        String time =
jsonArray.getJSONObject(progress_value).getString("Time");
                        String kpcc =
jsonArray.getJSONObject(progress_value).getString("KPCC");
```

```
                    String density =
jsonArray.getJSONObject(progress_value).getString("Density");

                    arrayList.add(time + "\n" + kpcc + "\n" + density);
                    adapter.notifyDataSetChanged();
                } catch (JSONException e) {
                    Log.e("Error", e.toString());
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

        }
    });

    requestQueue.add(jsonArrayRequest);
  }
}

public void seebbarr( ){
  seek_bar = (SeekBar)findViewById(R.id.seekBar);
  text_view =(TextView)findViewById(R.id.textView);
  text_view.setText("Covered : " + seek_bar.getProgress() + " / " +seek_bar.getMax());

  seek_bar.setOnSeekBarChangeListener(
        new SeekBar.OnSeekBarChangeListener() {



          @Override
```

```
                public void onProgressChanged(SeekBar seekBar, int progress, boolean
fromUser) {
                    progress_value = progress;
                    text_view.setText("Covered : " + progress + " / " +seek_bar.getMax());
                    Toast.makeText(MainActivity.this,"SeekBar in
progress",Toast.LENGTH_LONG).show();
                }

                @Override
                public void onStartTrackingTouch(SeekBar seekBar) {
                    Toast.makeText(MainActivity.this,"SeekBar in
StartTracking",Toast.LENGTH_LONG).show();
                }

                @Override
                public void onStopTrackingTouch(SeekBar seekBar) {
                    text_view.setText("Covered : " + progress_value + " / " +seek_bar.getMax());
                    Toast.makeText(MainActivity.this,"SeekBar in
StopTracking",Toast.LENGTH_LONG).show();
                }
            }
        );

    }
}
```

m. Android Maps Activity

```
package lict.nsu.volleyexample;

import android.graphics.Color;
import android.support.v4.app.FragmentActivity;
import android.os.Bundle;
import android.widget.TextView;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {

    private GoogleMap mMap;

    String passedVar=null;
    private TextView passedView=null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_maps);
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager()
                .findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
        //Get our passed variable from our intent's EXTRAS
        passedVar=getIntent().getStringExtra(MainActivity.ID_EXTRA);
        //find out text view
        /*passedView=(TextView)findViewById(R.id.textView);
        //display our passed variable!!!
        passedView.setText("YOU CLICKED ITEM ID="+passedVar);*/
        changingcolour();
    }


    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera. In this
case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once the user has
     * installed Google Play services and returned to the app.
     */
    @Override
    public void onMapReady(GoogleMap googleMap) {
        mMap = googleMap;

        // Add a marker in Sydney and move the camera
        LatLng Thesis = new LatLng(23.780383, 90.407521);
        mMap.addMarker(new MarkerOptions().position(Thesis).title("Point of data
Collection"));
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(Thesis, 18));

        Polyline polyline1 = googleMap.addPolyline(new PolylineOptions()
                .clickable(true)
                .add(
                        new LatLng(23.780493, 90.406935),
                        new LatLng(23.780382, 90.407554),
                        new LatLng(23.780316, 90.409366))
                .width(5)
                .color(Color.BLUE));

        polyline1.setTag("progress_value");




    }

    /*public void changingcolour(){


        if (passedVar>=25){
            Polyline polyline1 = PolylineOptions().color(Color.BLUE);
        }
        if (passedVar<=36){
            Polyline polyline1 = PolylineOptions().color(Color.RED);
        }
    }*/

                /*PolylineOptions rectOptions = new PolylineOptions();




        @Override
        public void onLocationChanged(Location location) {
            //   mMessageView.setText("Location = " + location);

            rectOptions.add(new LatLng(location.getLatitude(), location.getLongitude()));
```

```
            if (setIt == true){
                setcolortopoly();
                mMap.addPolyline(rectOptions);
            }
        }

        public void setcolortopoly(){
            if(mIsMetric){
                if(speed_poly<5.0){
                    rectOptions = new PolylineOptions().width(3).color(
                            Color.YELLOW );
                }
                else if(5.0>= speed_poly && speed_poly <10){
                    rectOptions = new PolylineOptions().width(3).color(
                            Color.GREEN );
                }
                else if(10.0>=speed_poly){
                    rectOptions = new PolylineOptions().width(3).color(
                            Color.RED );
                }

            }
            else{
                if(speed_poly<3.1){
                    rectOptions = new PolylineOptions().width(3).color(
                            Color.YELLOW );
                }*/
}
```

n. Android Main Activity Layout

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="10dp">


    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="88dp"
        android:scrollbars="horizontal"
        android:visibility="visible"></ListView>

            <SeekBar
                android:id="@+id/seekBar"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_alignParentEnd="true"
                android:layout_alignParentTop="true"
                android:layout_marginTop="13dp" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/listView"
        android:layout_marginTop="62dp"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <RadioGroup
        android:id="@+id/days"
        android:layout_width="350dp"
        android:layout_height="60dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="132dp"
        android:orientation="horizontal">

        <RadioButton
            android:id="@+id/radioButton7"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Sat" />

        <RadioButton
            android:id="@+id/radioButton6"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Frid" />

        <RadioButton
            android:id="@+id/radioButton5"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Thur" />

        <RadioButton
            android:id="@+id/radioButton4"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Wed" />

        <RadioButton
            android:id="@+id/radioButton3"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Tues" />

        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Mon" />

        <RadioButton
            android:id="@+id/radioButton1"
```

```
            android:layout_width="50dp"
            android:layout_height="50dp"
            android:layout_gravity="bottom"
            android:checked="true"
            android:text="Sun" />
    </RadioGroup>

    <Button
        android:id="@+id/day_sbm"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentStart="true"
        android:layout_marginBottom="22dp"
        android:layout_marginStart="42dp"
        android:text="choose day" />

    <Button
        android:id="@+id/go2map"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignTop="@+id/day_sbm"
        android:layout_marginEnd="59dp"
        android:text="Go to Map" />

</RelativeLayout>
```