KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY KUMASI
SCHOOL OF GRADUATE STUDIES

DEPARTMENT OF TELECOMMUNICATIONS ENGINEERING

COLLEGE OF ENGINEERING

DESIGN OF A NETWORK TRAFFIC PREDICTION MODEL USING THE KALMAN FILTER

BY

ODURO-AFRIYIE JOEL

MARCH, 2014

DESIGN OF A NETWORK TRAFFIC PREDICTION MODEL USING THE KALMAN FILTER

By

ODURO-AFRIYIE JOEL

A Thesis submitted to the

DEPARTMENT OF TELECOMMUNICATIONS ENGINEERING,

Kwame Nkrumah University of Science and Technology,

in partial fulfilment of the requirement for the degree of

MASTER OF SCIENCE

COLLEGE OF ENGINEERING

March, 2014

DECLARATION

I hereby declare that, except for specific references which have been duly acknowledged, this work is the result of my own field research and it has not been submitted either in part or whole for any other degree elsewhere.

Signature…… Date.....

ODURO-AFRIYIE Joel

(Candidate)

Signature…….. Date.....

Dr David ANIPA

(Supervisor)

Signature…….. Date …….

Dr P.Y. OKYERE

(Head, Department of Electrical / Electronic Engineering)

ABSTRACT

Network traffic prediction is of immense interest to industry since it supports decision making in management and control, and eventually user satisfaction. In this project, a Kalman filter-based model was developed for predicting network traffic. The focus was on telecommunication, transportation and computer networks. Traffic volume per unit time, or traffic flow rate, observed in a particular time interval, was utilized to predict the traffic flow rate for the next time interval by recursively computing relevant parameters of incoming traffic data. Relevant parameters include the process and measurement noise covariances, the Kalman filter gain and the a-priori and a-posteriori state and covariance estimates. The model makes use of the Kalman filter to carry out the prediction, and was tested using traffic sets with low, average and high autocorrelation. By means of a LabVIEW VI (simulation tool) different parameters were varied and their effects on the prediction model observed. LabVIEW was employed for its superior simulation features, with an integrated MATLAB block for optimization. Working with a 20% error tolerance, prediction accuracies approached 90%, and this process yielded an improved short-term traffic flow rate prediction model. The model carried out prediction for a single time-step ahead but, with refinements or modifications, it may be employed for multi-step prediction, converting it to a long-term predictor.

ACKNOWLEDGEMENTS

I would like to appreciate my supervisor, Dr. David K. Anipa, for his great patience, encouragement, direction and insight throughout this project. His forward thinking and dedicated mentoring have been an invaluable source of inspiration in this project.

Much appreciation goes to Dr. James Gadze, whose very professional guidance and assistance were priceless during certain periods when my supervisor Dr. Anipa could not be available.

My thanks also go to Peter D. Joseph, whose very lucid materials on the Kalman filter were a great help in my research into Kalman filter theory. His ready response to my inquiries were also very much appreciated.

I am deeply grateful to Joel Hesch of the University of Minnesota, who was incredibly welcoming and forthcoming with help on Kalman filter theory when approached.

To Greg Welch and Gary Bishop of the University of North Carolina, who did not hesitate to provide direction when contacted, I am very thankful.

I am indebted to Tom Lane of The Mathworks®, whose code examples and patient responses to my mails were priceless components in developing the MATLAB aspect of this project.

My sincere gratitude goes also to James Baffoe of Millicom Ghana for his never-ending readiness to provide mathematical and statistical insight whenever I needed it.

I owe a depth of gratitude to my colleague Strignner Bedu-Addo, who gave me my first LabVIEW tutorial and thus opened up to me the immense potential of the software.

To all my Telecom Engineering course mates, I am eternally grateful for the constantly willing input and contributions to problems pertaining to this project.

To all friends (including Naa Norkor Nartey and Miriam Ohene-Okantah) who showed interest in the project, and who gave valuable opinions and input, I wish to extend my deepest gratitude.

Finally, and most importantly, my utmost appreciation goes to God Almighty, without whose enablement and constant strengthening this project would never have taken off in the first place. I owe every success and achievement in this project to Him.

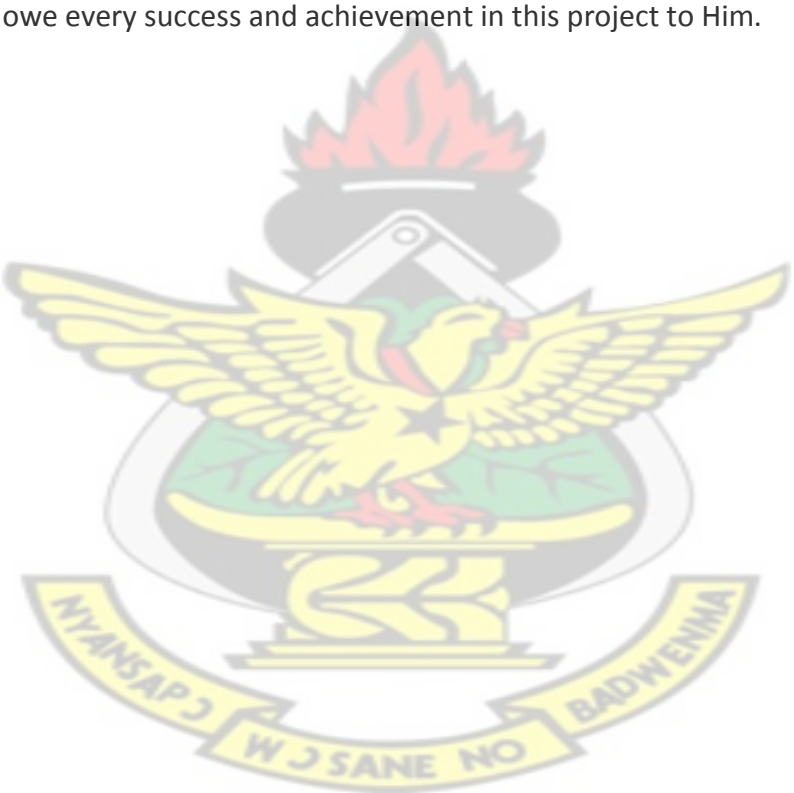


TABLE OF CONTENTS

INTRODUCTION

1.1 Background and Motivation

The practice of predicting network traffic is gaining more and more popularity as the importance becomes more and more apparent. Communications networks are becoming much more efficient as they employ traffic prediction concepts. Transportation network planners are receiving a boost in congestion management as a result of traffic prediction applications. For example, http://trafficpredict.com provides traffic prediction services for road users in the city of Los Angeles in the United States.

The network and traffic efficiencies reaped from such applications tend spill over into other areas of national economy. Productivity increases as a result of savings in time and cost, as well as general satisfaction of citizens.

It is no surprise therefore that many methods have been developed over the years for the purpose of data forecasting. These include multiple regression analysis, nonlinear regression, trend analysis, decomposition analysis, moving average analysis, weighted moving averages, adaptive filtering, exponential smoothing, the Hodrick-Prescott filter[1] and the Kalman filter, to name a few. Each of these methods possesses its merits and demerits, and some methods are more suited to particular applications such as traffic flow rate forecasting.

However, considering that developing countries generally experience more economic struggles, it is developing countries that stand to gain the most from such technologies. As already discussed in the previous section, the ripple effects of better traffic management on economies could go a long way to ease their economic burdens.

This makes it rather expedient that such a technology be developed that would be affordable and useful to a developing country such as Ghana.

1.2 **Problem Statement**

Most of the currently existing traffic forecasting systems are quite complicated and therefore rather costly. They are typically funded by thousands of dollars of government sponsorship. However, many developing countries are not yet at a position where huge government investments are made in such technologies.

In order for developing countries like Ghana to enjoy the benefits of traffic prediction applications, such as are outlined in the previous section, such applications will have to be quite affordable without sacrificing the efficiency of the system. Granted that more investment may be required to develop more accurate and efficient systems, a reasonable amount of efficiency can be achieved at a reasonable cost.

This project is therefore focused on developing a cost-effective but reasonably efficient traffic prediction system. Owing to cost considerations, the project considers short-term forecasting, with the potential for modification and application to long-term forecasting through further research.

1.3 **General Objectives**

The above-stated goal will be achieved as follows:

a) Develop a traffic flow rate forecasting model based on the Kalman Filter:

   i. *Describe the traffic flow as a system of equations*

   - This step will produce a linear combination representing the traffic flow

ii. *Model the system of equations as a Kalman filter*

- This step will produce a form of the equation in (i) above to which the Kalman filter can be applied.

b) Implement short-term traffic flow rate forecasting using the model developed

i. *Use the Kalman filter model to estimate/predict the traffic flow rate for time t+1, using n previous traffic values up to time t:*

- This will produce an equation for predicting the traffic flow rate for the next time step.
- By means of this equation, the traffic flow rate estimate for the next time step will be obtained.

ii. *Apply this process recursively to achieve real-time short-term traffic prediction*

1.4 **Thesis Organization**

This thesis is organized as follows:

Chapter 1 presents a general overview of the project, covering the motivation for the project, and general objective, amongst others.

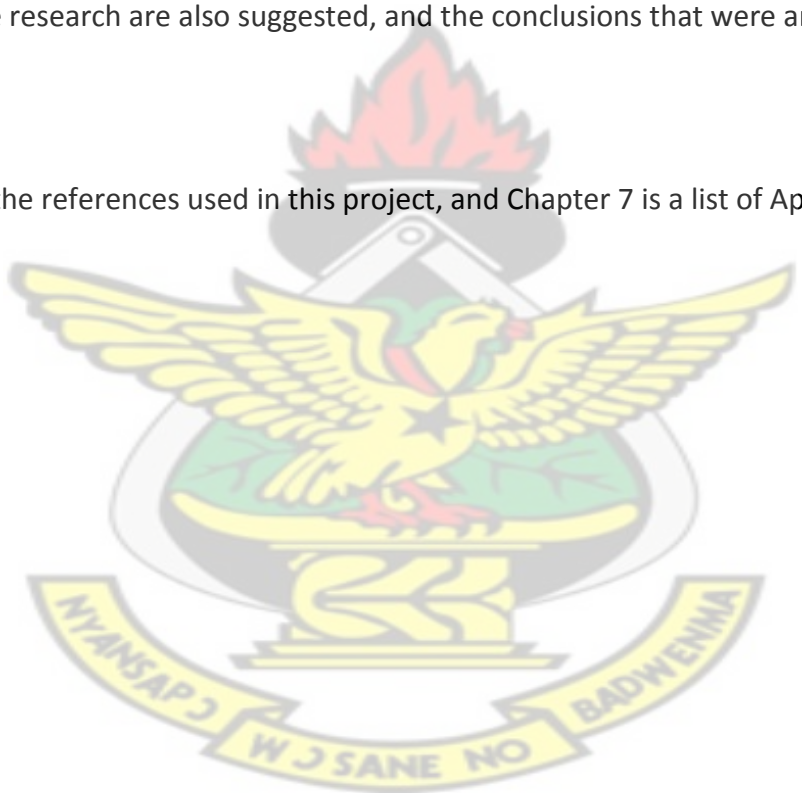
This is followed by a treatment of the Kalman filter in Chapter 2. The general Kalman filter theory is presented, along with the filter equations. This chapter attempts to induce an intuitive grasp of the operation of the Kalman filter. It also presents various examples of previous projects that have made use of the Kalman filter for traffic forecasting.

Chapter 3 deals with the methodology employed to achieve the stated goals and objectives. It presents an analysis of the problem, and the models and software code that were employed in order to implement the desired traffic flow rate prediction. It also details the testing scenarios and processes.

A description of the results obtained from the testing phase is presented in Chapter 4. The data used, results obtained, and analysis and deductions are all described in this chapter.

This is followed by Chapter 5 where a summary of the work done is presented. Possible areas for future research are also suggested, and the conclusions that were arrived at are given.

Chapter 6 lists the references used in this project, and Chapter 7 is a list of Appendices.



2. OVERVIEW OF KALMAN FILTERS

2.1 Introduction

Most practical engineering problems require the estimation of parameters associated with physical phenomenon based on inaccurate measurements. Many algorithms exist today for parameter estimation, but the Kalman filter stands out as one of the best of such tools and is employed in many engineering processes that can be described by a linear system. In mathematical terms, the Kalman filter (KF) estimates the states of a linear system. The KF not only works in practice but is theoretically attractive as it is able to minimize the variance of the estimated error. It can be described as an optimal linear estimator.

In this chapter, the salient features of the KF that relate to this project are presented.

2.1.1 Kalman Filter Theory and Algorithm

The Kalman filter was named after Rudolf Emil Kalman, who first introduced the filter in 1960. The filter has been employed in a myriad of applications including process control systems, vehicle tracking, marine navigation, geology, demographic estimation and stock price prediction. The filter estimates the instantaneous state of a linear dynamic system perturbed by Gaussian white noise by using measurements that are linearly related to the system state but that are corrupted by Gaussian white noise.

The filter recursively minimizes the mean square estimation error without directly observing the system state or knowing the nature of the modeled system. Since the time of its introduction, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. This is likely due in large part to advances in digital computing that made the use of the filter practically, but also to the relative simplicity and robust nature of the filter itself. Rarely do the conditions necessary

for optimality actually exist, and yet the filter apparently works well for many applications in spite of this situation. Kalman described his filter using state space techniques, which enables the filter to be used as a smoother, a filter or a predictor. The predicting ability of the filter is what this project seeks to make use of.

The Kalman filter addresses the problem of attempting to estimate the state of a discrete-time controlled process. The state is represented by two variables:

- $\hat{x}_{k/k}$, the estimate of the state at time k given observations up to and including time k;
- $P_{k/k}$, the error covariance matrix (a measure of the accuracy of the state estimate).

Discrete-time linear systems are often represented in a state-variable format given by a state equation:

$$x_k = ax_{k-1} + bu_k + w_k ----(2.10)$$

and a measurement equation:

$$z_k = hx_k + v_k ------- (2.11)$$

where the state x_k is a scalar, a and b are constants and the input u_k is a scalar; *k* represents the time variable. The noise w_k is a white noise source with zero mean and covariance Q that is uncorrelated with the input. Likewise, the noise v_k is a white noise source with zero mean and covariance R that is uncorrelated with the input. The input u_k usually defaults to zero, so it is sometimes omitted. The two equations above (equations 1 and 2) form the basis of the Kalman filter algorithm.

2.1.2 Kalman Filter Equations

The Kalman filter is essentially a set of mathematical equations that implement a predictor-corrector type estimator that is optimal in the sense that it minimizes the estimated error covariance when some presumed conditions are met.

The filter has two distinct stages: the time update (Predictor) and the measurement update (Corrector) stages.

The Time Update (Predictor) Equations

At this stage of the Kalman filter algorithm, the state of the process x_k under investigation is projected or predicted. This initial estimate is called the a priori estimate $\hat{x}_k^-$.

The a priori estimate $\hat{x}_k^-$ is then used to predict an estimate for the output $\hat{z}_k$. The difference between the estimated output and the actual output, called the residual or innovation, is then computed using equation 2.12 (with the help of equation 2.11):

$$residual = z_k - \hat{z}_k = z_k - h\hat{x}_k^- \quad ----- \quad (2.12)$$

The residual is then used to refine the initial estimate for the state x_k to obtain a new estimate called the a posteriori estimate, $\hat{x}_k$

$$\hat{x}_k = \hat{x}_k^- + K(residual) = \hat{x}_k^- + K(z_k - h\hat{x}_k^-) \quad ----- \quad (2.13)$$

Where K is the Kalman gain. The measurement update (corrector) equations are then used to correct the estimates of the time update stage.

The Measurement Update (Corrector) Equations

At this stage of the algorithm, the Kalman gain K is first computed. The computed gain is then used to update the a posterior estimate via the output z_k . The error covariance is finally updated.

2.1.3 Derivation of the Kalman Gain

To begin, the two errors of the estimate, the a priori error e_k^- and the a posteriori error e_k, are defined. The a priori and a posteriori errors are defined as the difference between the actual value of x_k and the a priori and a posteriori estimates respectively.

$$\left.\begin{matrix} e_k^- = x_k - \hat{x}_k^- \\ e_k = x_k - \hat{x}_k \end{matrix}\right\} - - - - - \quad (2.14)$$

The a priori and the a posteriori errors, are each associated with mean squared errors or variances represented in the equation 2.15 below.

$$\left.\begin{matrix} p_k^- = E\{(e_k^-)^2\} \\ p_k = E\{(e_k)^2\} \end{matrix}\right\} - - - - - - (2.15)$$

To start off, equation 2.13 is substituted into equation 2.14, and the resultant equation is finally substituted into equation 2.12, yielding equation 2.16.

$$p_k = E\{(x_k - \hat{x}_k)^2\} p_k = E\{(x_k - \hat{x}_k^- + K(z_k - h\hat{x}_k^-))^2\} - - - - - (2.16)$$

In order to find the value of K, the expression obtained in equation 2.16 is differentiated with respect to K, and the derivative is set to zero:

$$\frac{\partial p_k}{\partial k} = 0 = \frac{\partial E\left\{\left(x_k - \hat{x}_k^- + K(z_k - h\hat{x}_k^-)\right)^2\right\}}{\partial k}$$

$$= 2E\left\{\left(x_k - \hat{x}_k^- + K(z_k - h\hat{x}_k^-)\right)(z_k - h\hat{x}_k^-)\right\}$$

$$= 2E\left\{x_k z_k - \hat{x}_k^- z_k + K z_k^2 - Kh\hat{x}_k^- z_k - h x_k \hat{x}_k^- + (\hat{x}_k^-)^2 - Khz_k\hat{x}_k^- + Kh^2(\hat{x}_k^-)^2\right\} - -(2.17)$$

Working through the expression for K yields equation 2.18:

$$K = \frac{E\{x_k z_k - \hat{x}_k^- z_k - h x_k \hat{x}_k^- + h(\hat{x}_k^-)^2\}}{E\{z_k^2 - 2h\hat{x}_k^- z_k + h^2(\hat{x}_k^-)^2\}} - - - - - (2.18)$$

To overcome the cumbersome nature of this expression, the numerator and the denominator are treated separately.

From the basic assumption that the measurement noise v is uncorrelated to either the input or the a priori estimate of x, it follows that

$$E\{(x_k - \hat{x}_k^-)v_k\} = E\{e_k^- v_k\} = 0 - - - - - (2.19)$$

This simplifies the expression for the numerator to:

$$\begin{aligned} numerator &= E\{h x_k^2 - 2h x_k \hat{x}_k^- + h(\hat{x}_k^-)^2\} \\ &= hE\{(x_k - \hat{x}_k^-)^2\} = E\{(e_k^-)^2\} \\ &= hp_k^- \ - - - - - (2.20) \end{aligned}$$

Using the orthogonal condition for the denominator of the expression in equation 2.18, the denominator evaluates to equation 2.21 below:

$$\begin{aligned} denominator &= E\{h^2 x_k^2 - 2h^2 \hat{x}_k^- x_k + h^2(\hat{x}_k^-)^2 + v_k^2\} \\ &= h^2 E\{x_k^2 - 2\hat{x}_k^- x_k + (\hat{x}_k^-)^2\} + E\{v_k^2\} \\ &= h^2 p_k^- + R - - - - - (2.21) \end{aligned}$$

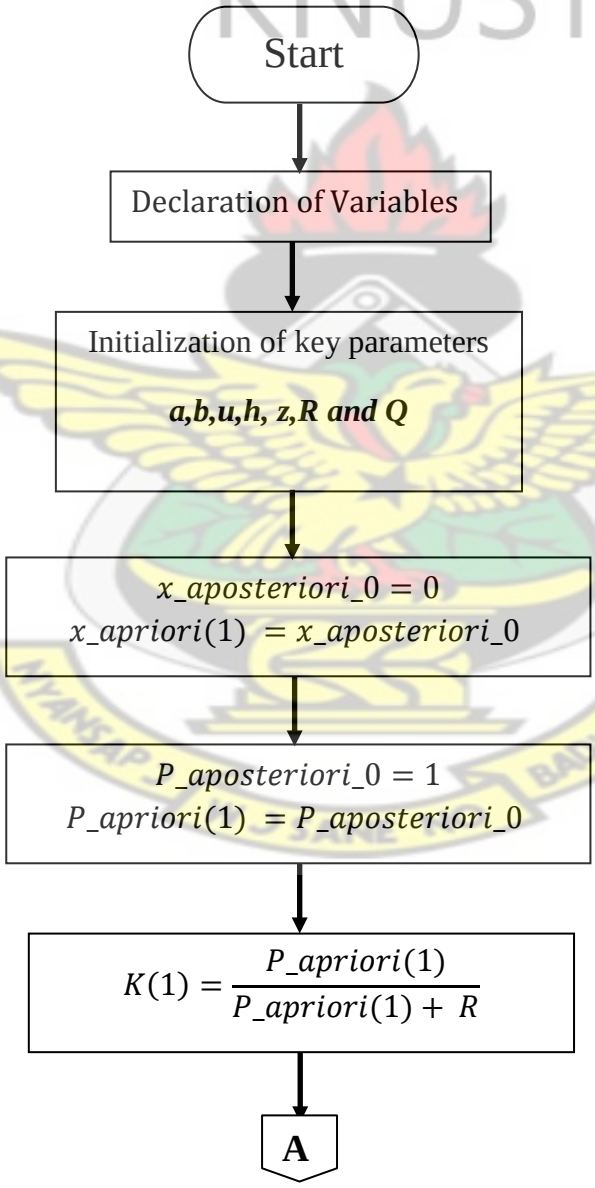
Substituting the new expressions of the numerator and the denominator into equation 2.18 yields the simplified equation for the Kalman gain:

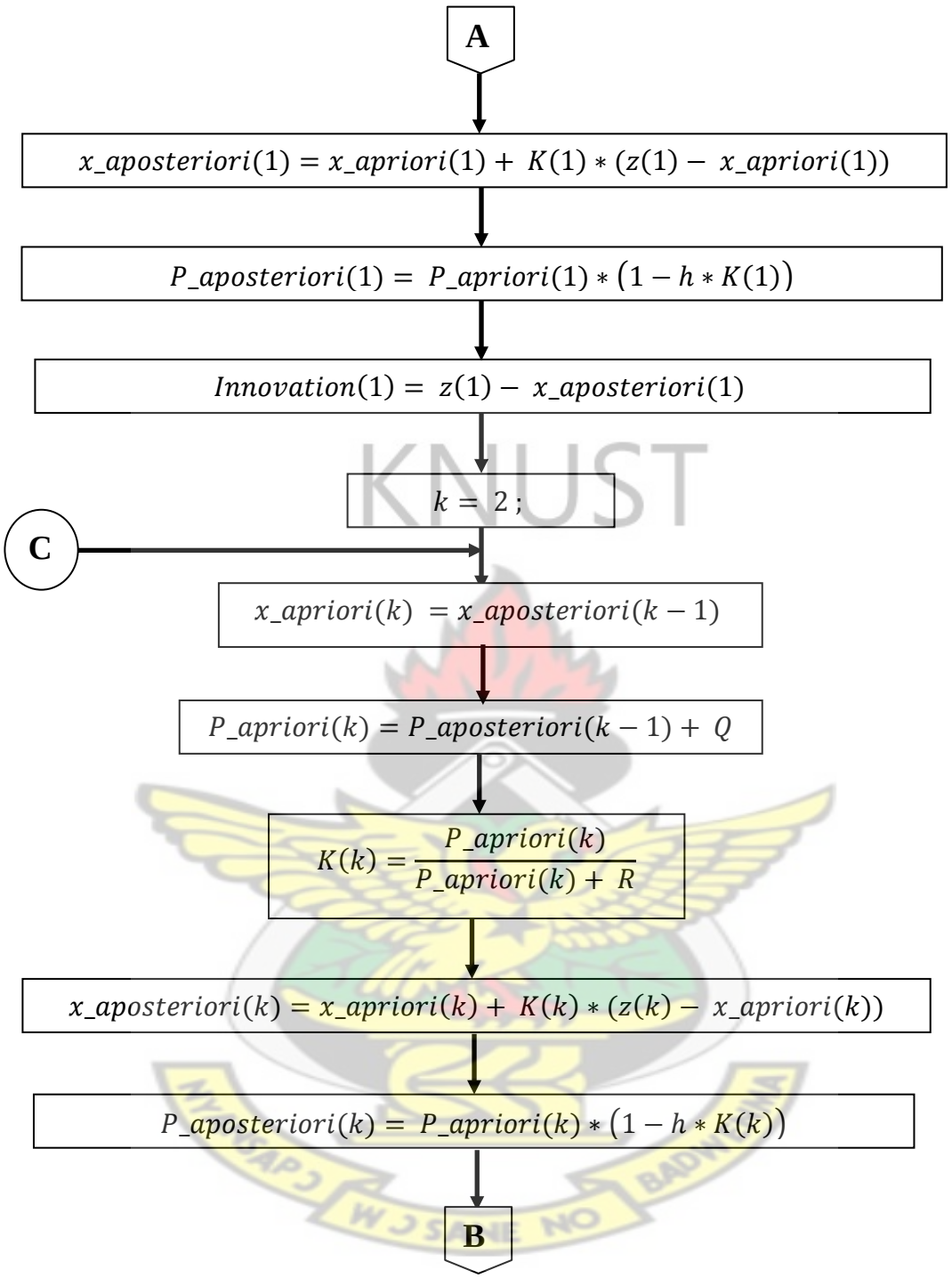
$$K = \frac{hp_k^-}{h^2 p_k^- + R} - - - - - (2.22)$$

Similar techniques used for the derivation of the Kalman gain can be employed to derive the covariance errors of the estimation.

Detailed Flowchart for the Kalman Filter Algorithm

In this section, a detailed flowchart of the Kalman filter algorithm is presented. This can easily be translated into a computer program using any relevant programming language.




A
$x\_aposteriori(1) = x\_apriori(1) + K(1) * (z(1) - x\_apriori(1))$
$P\_aposteriori(1) = P\_apriori(1) * (1 - h * K(1))$
$Innovation(1) = z(1) - x\_aposteriori(1)$
$k = 2;$
C
$x\_apriori(k) = x\_aposteriori(k-1)$
$P\_apriori(k) = P\_aposteriori(k-1) + Q$
$K(k) = \frac{P\_apriori(k)}{P\_apriori(k) + R}$
$x\_aposteriori(k) = x\_apriori(k) + K(k) * (z(k) - x\_apriori(k))$
$P\_aposteriori(k) = P\_apriori(k) * (1 - h * K(k))$
B

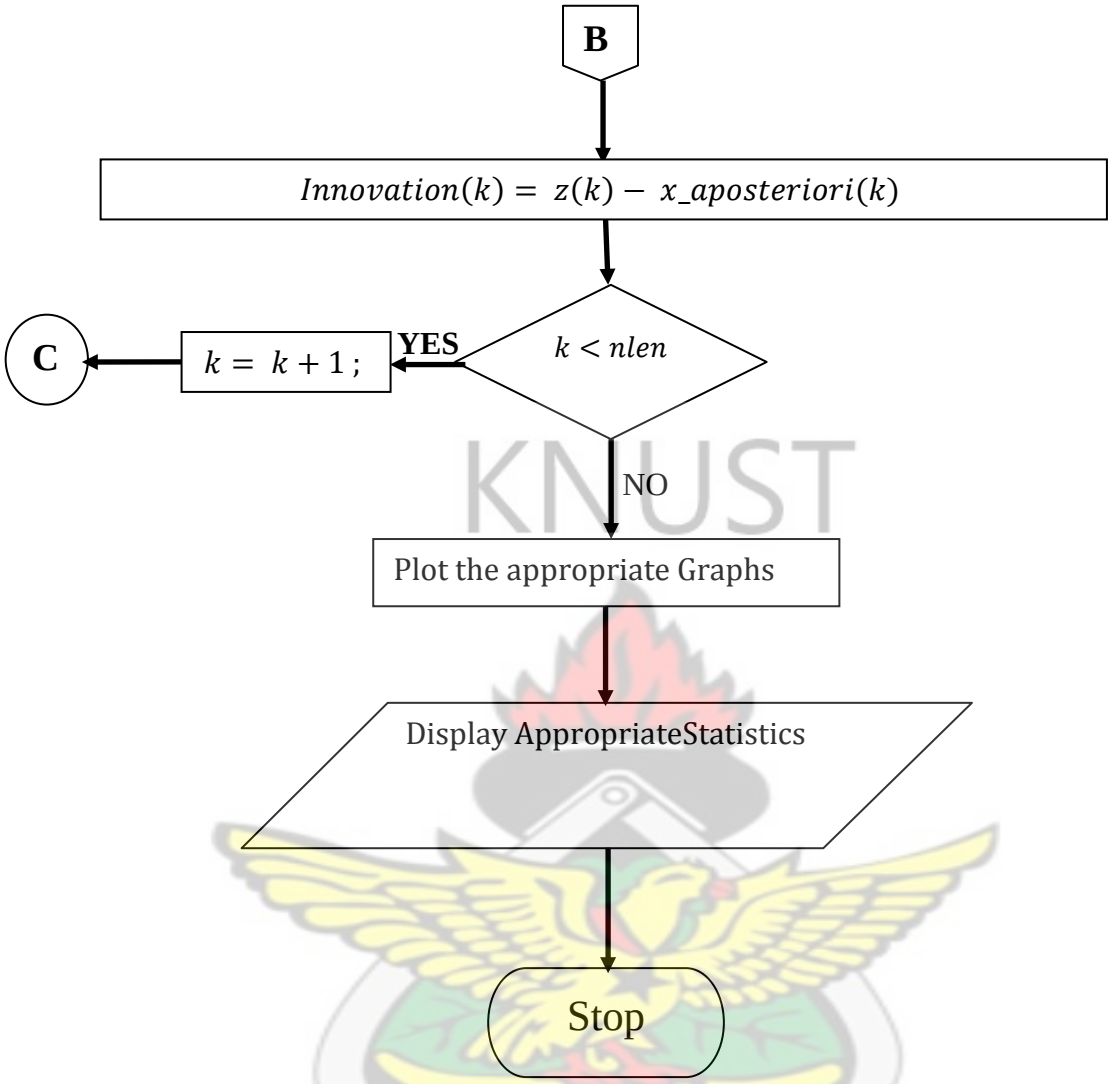


Fig. 2.1: Kalman Filter Flowchart

2.1.4 A Step – by – Step Walkthrough of the Kalman Filter

In this section a step-by-step approach in explaining the Kalman filter theory is presented.

Step 1 – Building a Model

At this stage, it is important to first of all determine that Kalman filtering conditions fit the problem under investigation.

Equations 2.10 and 2.11 are reproduced here for convenience, to serve as a guide:

$$x_k = Ax_{k-1} + Bu_k + w_{k-1} \text{ ---------- (2.10)}$$

$$z_k = Hx_k + v_k \text{ ---------- (2.11)}$$

From equation 2.10, each x_k (our signal values) may be evaluated by using a linear stochastic equation as shown above. Any x_k is a linear combination of its previous value plus a control signal u_k and a process noise w_k. According to equation 2.11, any measurement value (whose accuracy is uncertain) is a linear combination of the signal value and the measurement noise v_k. The process noise and the measurement noise are statistically independent. The entities A, B and H are in general form matrices. For the sake of this discussion, they will be assumed to be numerical constants.

Step 2 – Starting the Process

Once the model built at step 1 fits into the Kalman filter equations, the next step is to determine the necessary parameters and initial values. At this stage, the time update (predictor) and the measurement update (corrector) equations come in useful. Both equations are applied at each k^{th} state of the process.

Time Update (prediction)	Measurement Update (correction)
$\hat{x}_k = A\hat{x}_{k-1} + Bu_k$ $P_k^- = AP_{k-1}A^T + Q$	$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1}$ $\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$ $P_k = (I - K_k H)P_k^-$

Table 2.1: Kalman Filter Stages

Based on the assumption that A, B and H are numerical constants and, for this discussion, setting each of their values to 1, Q and R may be determined. $Q$and R are variances of the process and measurement errors respectively. R is rather easy to find because, in general, the noise in the environment is known. But finding Q is not so obvious. To start the process effectively, the initial estimates of x_0 and P_0 have to be assumed.

Step 3 – Iteration

When all the relevant information is gathered and the process has started, iteration through the estimates can begin. It is important to note here that the previous estimates will be the input for the current state. The whole iteration process is illustrated below:

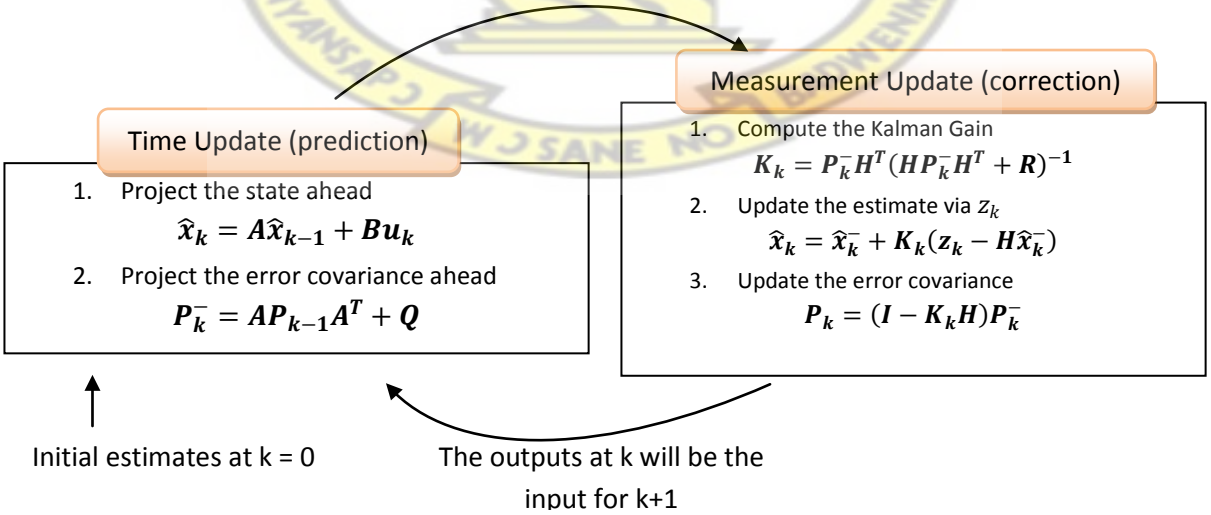


Fig 2.2: Kalman Filter iterative process

Here $\hat{x}_k^-$ is the a priori estimate, which stands for the rough estimate before the correction. Also P_k^- is the a priori error covariance. These a priori values will be used in the measurement update equations. In the measurement update equations,$\hat{x}_k$, which is the estimate of x at time k, is computed. Also, P_k, which is necessary for estimates at next the k step, is calculated. The Kalman gain K_k is also evaluated at this stage. The values evaluated at the measurement update stage are also called a posteriori values.

2.2 Related Work (Use of Kalman filters in network traffic prediction)

The Kalman filter has been used in many previous research efforts for traffic prediction. Below, a few example applications are highlighted.

2.2.1 Short-Term Traffic Volume Forecasting Using Kalman Filter with Discrete Wavelet Decomposition

Yuanchang Xie, Yunlong Zhang and Zhirui Ye applied the Kalman filter in traffic flow rate forecasting, in their paper "Short-Term Traffic Volume Forecasting Using Kalman Filter with Discrete Wavelet Decomposition" [2].

In their work, they investigated the application of a Kalman filter with discrete wavelet analysis in short-term traffic flow rate forecasting. Short-term traffic flow rate data are often corrupted by local noises, which may significantly affect the prediction accuracy of short-term traffic flow rates. Therefore, they used discrete wavelet decomposition analysis to divide the original data into several approximate and detailed data such that the Kalman filter model could then be applied to the de-noised data to improve the prediction accuracy.

This method, though accurate, demands a lot of processing power, and is expensive.

2.2.2 **Short-term traffic safety forecasting using Gaussian mixture model and Kalman filter** [11]

In this work by Sheng Jin, Dian-hai Wang, Cheng Xu and Dong-fang Ma, a prediction model is developed that combines a Gaussian mixture model (GMM) and a Kalman filter for online forecasting of traffic safety on expressways. Raw time-to-collision (TTC) samples are divided into two categories: those representing vehicles in risky situations and those in safe situations. Then, the GMM is used to model a bimodal distribution of the TTC samples, and the maximum likelihood (ML) estimation parameters of the TTC distribution are obtained using the expectation-maximization (EM) algorithm. They propose a new traffic safety indicator, which they call the proportion of exposure to traffic conflicts (PETTC), for assessing the risk and predicting the safety of expressway traffic. A Kalman filter is applied to forecast the short-term safety indicator, PETTC, and solves the online safety prediction problem. A dataset collected from four different expressway locations is used for performance estimation. [11]

This is an accurate method. However, it is expensive, and has an associated time and data cost for training the model.

2.2.3 **Kalman filter approach to traffic modeling and prediction**

Gregory J. Grindey, S. M. Amin, Ervin Y. Rodin and Asdrubal Garcia-Ortiz carried out this work with the aim of developing and integrating prediction, control and optimization modules for use in highway traffic management. This they accomplished through the use of the Semantic Control paradigm, implementing a hybrid prediction/routing/control system, to model both macro-level and micro level. Their paper addressed the design

and operation of a Kalman filter that processes traffic sensor data in order to model and predict highway traffic flow rate. This data was given in the form of hourly traffic flow, and a cubic spline method was used to fit the data to allow observations at various time intervals. The filter was augmented via the Method of Sage and Husa [20] to identify the parameters of the system noise on-line, and to determine the dynamics of the traffic process iteratively to aid in the prediction of the future traffic. [12]

Though accurate, this is an expensive method.

2.2.4 **Tracking and predicting a network traffic process**

Joe Whittaker, Simon Garsidea and Karel Lindveld presented a forecast system in 1997 that applied a Kalman filter on a motorway network around Rotterdam in the Netherlands to tackle the problem of real-time modelling and prediction of motorway traffic. They proposed conditional independence relationships and ideas of Bayesian forecasting [21] leading to the employment of dynamic state-space models, with optimal state estimation coming from the Kalman filter. They implemented and derived models in a state-space framework based on classical differential equations, which incorporated representations of the network topology. [13]

This method yielded reasonably accurate and real-time prediction, but at high cost.

2.2.5 **Real-time freeway traffic state estimation based on extended Kalman filter: a general approach**

This study, carried out by Yibing Wang and Markos Papageorgiou, developed a general approach to the real-time estimation of the complete traffic state in freeway stretches based on the extended Kalman filter. First, they presented a general stochastic

macroscopic traffic flow model of freeway stretches, while proposing some simple formulae to model real-time traffic measurements. Second, the macroscopic traffic flow model along with the measurement model was organized in a compact state-space form, based on which a traffic state estimator was designed by use of the extended-Kalman-filtering method. While constructing the traffic state estimator, special attention was paid to the handling of the boundary conditions and unknown parameters of the macroscopic traffic flow model. [15]

This is another accurate but expensive approach.

2.2.6 An Extended Kalman Filter Application for Traffic State Estimation Using CTM with Implicit Mode Switching and Dynamic Parameters

Chris M.J. Tampère and L. H. Immers produced this work with two main objectives [14]:

a) to show how the cell transmission model (CTM) [22] can be included in the general extended Kalman filtering(EKF) framework of Wang & Papageorgiou [15]; the key issue here is to linearize the non-linear CTM model around its current state, which is done implicitly by introducing an appropriate formulation of CTM;
b) to show the capability of the combined CTM-EKF model to capture (rapid) changes of important modelling parameters like the capacity.

Using both real and simulated data, they illustrated the applicability of the model in a case study on a motorway.

This method can rapidly adapt to changes, but again it comes with an unattractive cost.

2.2.7 **DynaMIT: a simulation-based system for traffic prediction**

DynaMIT (Dynamic Network Assignment for the Management of Information to Travelers) is a real time dynamic traffic assignment system that was developed at the Massachusetts Institute of Technology (MIT) to provide traffic predictions and travel guidance.

DynaMIT generates prediction-based guidance with respect to departure time, pre-trip path and mode choice decisions and en-route path choice decisions. It supports both prescriptive and descriptive information. In order to guarantee the credibility of the information system, the guidance provided by DynaMIT is consistent, meaning that it corresponds to traffic conditions that most likely will be experienced by drivers. Hence, DynaMIT provides user-optimal guidance, which implies that users cannot find a path that they would prefer compared to the one they chose based on the provided information.

DynaMIT features a dynamic OD (origin-destination) estimation process based on a Kalman filtering algorithm and on an auto-regressive process (Ashok and Ben-Akiva, 1993) [19]. The auto-regressive process, captures the dynamic evolution in time of the state variables of the Kalman filter. It is calibrated off-line and constitutes an input to the real-time system. [18]

This is another accurate method, but the associated cost is prohibitive.

3. METHODOLOGY

The focus of this project was to develop a suitable Kalman filter model to carry out traffic flow rate prediction. The test data was taken from a telecommunications network, but the concepts and techniques apply to any network, including transportation networks. The following section seeks to outline in detail the process of developing a reasonable model to predict traffic flow rates.

3.1 Model Development

The analysis in this section is carried out first over a single network link, and then extended to cover multiple network links. The Kalman model for a single link is treated as a scalar model, after which extension to multiple links is treated by means of matrices.

Single Link Analysis

As already stated in the introduction of this report, this project dealt with short term traffic flow rate forecasting. This is important because it allowed for certain assumptions: [2]

(a) For short term forecasting, the state variable transitions may be regarded as a smooth process.

(b) A linear relationship may be assumed between traffic flow rates for the current time step and traffic flow rates for previous time steps.

From (b) it may be inferred that the traffic flow rate at any particular time step is a linear combination of the traffic flow rates at previous time steps. Put another way, the previous traffic flow rates, each multiplied by a corresponding coefficient, may be summed up to give the traffic flow rate for the current time step.

Thus, if $traf_k$ represents the traffic flow rate at time step k, and c_k represents the corresponding coefficient that multiplies $traf_k$, then from (b) it may be inferred that

$$traf_k = traf_{k-1}c_{k-1} + traf_{k-2}c_{k-2} + ... + traf_{k-n}c_{k-n} + v_k \qquad (3.10)$$

where v_k is the noise contribution at time step k and n represents the number of previous traffic flow rates taken into consideration.

The right-hand-side of (3.10) may be expressed in matrix form by defining

$$TRAF_k = [traf_{k-1}, traf_{k-2}, traf_{k-2},..., traf_{k-n}] \text{ and} \qquad (3.11a)$$

$$C_k = [c_{k-1}, c_{k-2}, c_{k-3},..., c_{k-n}]^T \qquad (3.11b)$$

then (3.10) may be written as

$$traf_k = TRAF_kC_k + v_k \qquad (3.12)$$

Comparing (3.11) to (2.11), it is observed that a suitable way to model the system is to set

$traf_k = z_k$, the measured output volume

$TRAF_k = h$, the output gain matrix, and

$C_k = x_k$, the system state

With the system thus modelled, the output equation of the filter (equation (2.11)) may be written as

$$\hat{traf}_k = TRAF_k \hat{C}_k^- \qquad (3.13)$$

where

$\hat{traf}_k$ represents the estimated output traffic flow rate, and

$\hat{C}_k^-$ represents the a priori state estimate.

After obtaining the estimated output traffic flow rate $\hat{traf}_k$, the *measured* output traffic flow rate $traf_k$ is received. $traf_k$ is used to update the output gain matrix $TRAF_k$ to obtain $TRAF_{k+1}$, i.e. $TRAF_{k+1} = [traf_k, traf_{k-1}, traf_{k-2}, \ldots, traf_{k-n+1}]$.

Equation (3.12) may then be used to achieve the filter's aim of predicting the output traffic flow rate for the next time step:

$$\hat{traf}_{k+1} = TRAF_{k+1}\hat{C}_k \qquad (3.14)$$

where $\hat{C}_k$ is the a posterior state estimate obtained according to equation (2.13).

Extension to multiple links

In order to extend the analysis of the previous section to cover multiple links, the network is treated as a matrix. The various parameters take on dimensions according to the number of links on the network and the number of previous traffic flow rates being considered.

For example, for a network of m links, an m by 1 output matrix may be defined to hold the output values for each of the m links. Likewise, an m by m measurement noise covariance matrix may be defined to hold the measurement noise covariance values for each of the m links. Also, for a filter that takes n previous traffic flow rates into consideration, an m by n output gain matrix may be defined to represent m output gain matrices each of length n to cater for the desired n previous traffic flow rates.

NOTATION:

To distinguish the parameters under the matrix Kalman system from their scalar counterparts, the following notational convention was employed:

- All matrix parameters have the same symbols as the scalar counterparts, but the matrix parameters are represented in bold face.

- All gain and covariance parameters for the matrix system are capitalized.

Table 3.1 summarizes the convention used to identify the parameters in the matrix Kalman filter system for this project, along with their respective dimensions. 'm' represents the number of links being considered in the network, and 'n' represents the number of previous traffic flow rates of interest.

Table 3.1[1]: Matrix Kalman filter variables and dimensions

Variable	Scalar	Matrix	Matrix size
state	x_k	$\mathbf{x}_k$	n by 1
input	u_k	$\mathbf{u}_k$	n by m
output	z_k	$\mathbf{z}_k$	m by 1
state gain	a	**A**	n by n
input gain	b	**B**	n by m
output gain	h	$\mathbf{H}_k$	m by n
process noise	w_k	$\mathbf{w}_k$	n by 1
process noise covariance	Q	**Q**	n by n
measurement noise	v_k	v_k	m by 1
measurement noise covariance	R	**R**	m by m
a priori covariance	p_k^-	$\mathbf{P}_k^-$	n by n
a posteriori covariance	p_k	$\mathbf{P}_k$	n by n
Kalman Filter Gain	k_k	$\mathbf{K}_k$	n by m

With the above notation, the Kalman filter equations may be rewritten for the matrix system thus:

[1]Adapted from http://www.swarthmore.edu/NatSci/echeeve1/Ref/Kalman/MatrixKalman.html

Matrix Kalman filter equations:

Predictor Stage

A priori state estimate:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} = \boldsymbol{A}\hat{\boldsymbol{x}}_{\boldsymbol{k-1}}^{-} + \boldsymbol{B}\boldsymbol{u}_{\boldsymbol{k}} - - - - - - - - \ (\mathbf{3.15})$$

A priori covariance:

$$\boldsymbol{P}_{\boldsymbol{k}}^{-} = \boldsymbol{A}\boldsymbol{P}_{\boldsymbol{k-1}}\boldsymbol{A}^{\boldsymbol{T}} + \boldsymbol{Q} - - - - - - - \ (\mathbf{3.16})$$

Corrector Stage

Kalman filter gain:

$$\boldsymbol{K}_{\boldsymbol{k}} = \boldsymbol{P}_{\boldsymbol{k}}^{-}\boldsymbol{H}^{\boldsymbol{T}}(\boldsymbol{H}\boldsymbol{P}_{\boldsymbol{k}}^{-}\boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{R})^{-\mathbf{1}} - - - - - - - - \ (\mathbf{3.17})$$

A posteriori state estimate:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}} = \hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} + \boldsymbol{K}_{\boldsymbol{k}}(\boldsymbol{z}_{\boldsymbol{k}} - \boldsymbol{H}\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}) - - - - - - - - \ (\mathbf{3.18})$$

A posteriori covariance:

$$\boldsymbol{P}_{\boldsymbol{k}} = (\boldsymbol{I} - \boldsymbol{K}_{\boldsymbol{k}}\boldsymbol{H})\boldsymbol{P}_{\boldsymbol{k}}^{-}(\boldsymbol{I} - \boldsymbol{K}_{\boldsymbol{k}}\boldsymbol{H})^{\boldsymbol{T}} + \boldsymbol{K}\boldsymbol{R}\boldsymbol{K}^{\boldsymbol{T}} - - (\mathbf{3.19})$$

where

$\hat{\boldsymbol{x}}$ = Estimated state.

A= State transition matrix (i.e., transition between states).

u = Control variables matrix.

B = Control matrix (i.e., mapping control to state variables).

P = State variance matrix (i.e., error of estimation).

Q = Process variance matrix (i.e., error due to process).

z = Measurement variable matrix.

H= Measurement matrix (i.e., mapping measurements onto state).

K= Kalman gain.

R= Measurement variance matrix (i.e., error from measurements).

I = Unit matrix

3.2 Model Implementation

Implementation of the model developed in section 3.1 was carried out by software in two forms:

i. Implementation with a Java program

ii. Implementation with LabVIEW

Apart from the fact that the Java programming language is well-known for its network-friendliness, the primary purpose of implementing the model in Java was to verify the correctness of the approach described for this filter implementation. The reason for carrying out the implementation in LabVIEW after the implementation in Java was basically to take advantage of the graphical user interface afforded by LabVIEW, where parameters can be varied in real time and the effects directly observed.

3.2.1 **Java program implementation**

The java program implementation was carried out simply by following the Kalman filter algorithm outlined in equations (3.15) to (3.19). The system was first initialized with the number m of links and the number n of previous traffic flow rates desired. Relevant parameters were initialized to their respective values, and the first set of data was fed to the system for the filter to begin operation.

The basic algorithm is outlined below:

a) Initialize the filter with m, the number of links on the network to be considered, and n, the number of previous traffic flow rates to be 'remembered' by the filter.

b) Initialize the filter's matrices to the right dimensions with m and n according to Table 3.1.

c) Initialize the matrices to their respective initial values.

d) Let k = 1, and carry out the Kalman filter algorithm (equations (3.15) to (3.19)):

   i) Compute the a priori state estimate and covariance $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$ and $\boldsymbol{P}_{\boldsymbol{k}}^{-}$ respectively, where $\hat{x}_k^-$ represents $\hat{C}_k^-$ (equations (3.13) and (3.11b)).

   ii) Compute the Kalman gain $\boldsymbol{K}_{\boldsymbol{k}}$ (equation (3.17)), where ***H*** represents $\boldsymbol{TRAF_k}$ (equation (3.11a)).

   iii) Obtain the measured traffic flow rate $traf_k$ for time step k, and use it to compute the a posteriori state estimate (equation (3.18)) and covariance (equation (3.19)), $\widehat{\boldsymbol{x}}_{\boldsymbol{k}}$ and $\boldsymbol{P}_{\boldsymbol{k}}$ respectively, where $\hat{x}_k$ represents $\hat{C}_k$ (equations (3.13) and (3.11b)).

   iv) Update $\boldsymbol{TRAF_k}$ with $traf_k$ to obtain $\boldsymbol{TRAF_{k+1}}$ (equation (3.11a)).

v) Compute the predicted traffic flow rate $\hat{traf}_{k+1}$ for the next time step k+1 according to equation (3.14), reproduced here for convenience:

$$\hat{traf}_{k+1} = TRAF_{k+1}\hat{C}_k$$

vi) Update the a priori state estimate and covariance to take on, respectively, the values of the a posteriori state estimate and covariance, i.e. let $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-} \leftarrow \hat{\boldsymbol{x}}_{\boldsymbol{k}}$ and $\boldsymbol{P}_{\boldsymbol{k}}^{-} \leftarrow \boldsymbol{P}_{\boldsymbol{k}}$.

vii) Let k = k+1, and go to i).

VARIABLE INITIALIZATION

Variable initialization has to do with assigning values to the parameters for time step zero (0), before the filter can carry out its first iteration. The various variables were initialized as follows:

State and input variables:

- The input variable matrix ***u*** was set to zero because the system takes in no input.
- The a priori state estimate $\hat{\boldsymbol{x}}_{\boldsymbol{k}}^{-}$, representing $\hat{C}_k^-$, was initialized to an n by 1 matrix of arbitrarily small values. In this project the value was set to 1/n. [2]

Gain variables:

- The state gain or state transition matrix ***A*** was set to be an n by n identity matrix because, from (a) of section 3.1, the state transition may be regarded as a smooth process for short term forecasting.
- The input gain matrix ***B*** was set to zero because, as already stated, the system takes no input.

- The output gain ***H***, representing ***TRAF$_k$***, was initially set to an m by n matrix of very small values, a value of 0.009 in the case of this project. This is the value used by the system as default traffic flow rate measurements before actual values are obtained.

Covariance variables:

- The measurement noise covariance matrix ***R*** was set to an m by m matrix with all entries initialized to0.015, a value obtained after averaging a collection of network traffic measuring equipment errors obtained from the internet.
- The a priori covariance $\boldsymbol{P_k^-}$ is customarily set to a matrix with very small values [2]. For this project a value of 0.009*$\boldsymbol{I_n}$ was used, where $\boldsymbol{I_n}$ is the n by n identity matrix.
- The process noise covariance ***Q*** may be obtained from the following log likelihood function [2]:

$$-\ln\big(L(Q)\big) = \sum_{k=1}^{n}\{\ln(X_k) + Y_k^T X_k^{-1} Y_k\} + C \quad ----- \quad (3.20)$$

where

$X_k = HP_k^- H^T + R$ (measurement prediction covariance)

$Y_k = z_k - H\hat{x}_k^-$ (measurement residual)

$P_k^- = AP_{k-1}A^T + Q$ (a priori covariance)

n = number of previous traffic flow rates 'remembered' by the system

C = constant

For a likelihood function, the aim is usually to maximize the function to obtain the most likely value. However, since the negative log is taken to facilitate computation, ***Q*** is obtained by *minimizing* rather than *maximizing* the function.

The MATLAB programming language is best suited for solving an optimization problem of the sort presented by the likelihood equation for **Q**. However, owing to the complexities involved in running MATLAB from within the Java environment, this equation was not implemented in Java. Thus, the matrix **Q** was initialized with arbitrary values.

Equation (3.20) was, however, evaluated in the LabVIEW implementation. As already stated, the Java implementation was carried out basically to verify the integrity of the chosen Kalman Filter model.

3.2.2 LabVIEW implementation

After verifying the model using the Java program, the model was implemented with LabVIEW. As already stated, the choice of LabVIEW for model implementation was to capitalize on the graphical and interactive nature of LabVIEW, where parameters can be varied, and changes observed, in real time.

The LabVIEW implementation was modelled after the same algorithm outlined under the Java program implementation.

VARIABLE INITIALIZATION

Variable initialization is simplified with the help of the in-built LabVIEW functions.

State, input and gain variables:

- Because the input variable matrix ***u*** was set to zero, (explained above), the LabVIEW implementation does not include this variable.
- The other variables were initialized to the values indicated in the 'Java Program Implementation' section. Here the LabVIEW 'Initialize Array' block was used. The parameters passed into this block were:
    - The value ('element' in Fig. 3 below) to initialize the matrix
    - The matrix dimensions

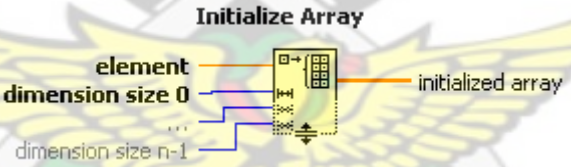


Fig 3.1: LabVIEW array initialization block

Covariance variables:

- The measurement noise covariance matrix ***R*** and the a priori covariance $\boldsymbol{P_k^-}$ were similarly initialized as for the state and gain variables.
- As shown in equation (3.20), reproduced here for convenience, the process noise covariance ***Q*** may be obtained by minimizing the following log likelihood function [2]:

$$-\ln\big(L(Q)\big) = \sum_{k=1}^{n}\{\ln(X_k) + Y_k^T X_k^{-1} Y_k\} + C \quad -----\quad (3.20)$$

where

$X_k = HP_k^- H^T + R$ (measurement prediction covariance)

$Y_k = z_k - H\hat{x}_k^-$ (measurement residual)

$P_k^- = AP_{k-1}A^T + Q$ (a priori covariance)

n = number of previous traffic flow rates 'remembered' by the system

C = constant

Note:

Likelihood functions generally attempt to find the value which gives the highest likelihood of the variable in question. This is usually achieved by ***maximizing*** *the likelihood function. However, for a* ***negative likelihood*** *function, the aim is to* ***minimize*** *the function in order to find the most likely value of the variable in question. Since equation (3.20) is a negative likelihood function, the most likely value of* ***Q*** *is obtained by minimizing equation (3.20).*

To evaluate this equation, it was necessary to employ the powerful optimization tools provided by the MATLAB programming language.

LabVIEW makes it possible to run MATLAB code directly from within the LabVIEW environment by means of the MATLAB script node. The MATALB script node accepts input from LabVIEW, runs whatever MATLAB code is typed in the node, and returns output to LabVIEW.

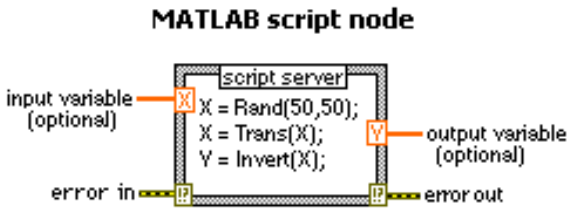


Fig 3.2: LabVIEW MATLAB script node

The MATLAB code below models equation (3.20):

Code Listing 3.0: MATLAB representation of equation (3.20)

```
Pk = 0;

nlogL = 0;

for k=1:n

Pk_ = A*Pk*A' + Q;

Yk = zk - H*xk_;

Xk = H*Pk_*H' + R;

nlogL = nlogL + log(Xk) + Yk'*inv(Xk)*Yk;

end
```

The constant C in equation (3.20) is taken to be zero.

The objective of equation (3.20) is to find the matrix **Q** which minimizes the log likelihood. That is, with respect to the MATLAB code above, we must find the **Q** which gives the smallest value of nlogL.

The MATLAB optimization toolbox presents a collection of functions which are well-suited to solving such a problem. The 'fminsearch' function proves to be suitable for this particular problem.

In order to use the 'fminsearch' function in MATLAB, it was necessary to rewrite code listing 3.0 as a MATLAB function that could be passed as a parameter to 'fminsearch'. The resulting function is shown below:

Code Listing 3.1: MATLAB function for equation (3.20)[9]

```
functionnlogL = qlogL(n,x,z,A,P,H,R,Q)
        Pk = 0;
        nlogL = 0;
        for k=1:n
        Pk_ = A*Pk*A' + Q;
        Yk = zk - H*xk_;
        Xk = H*Pk_*H' + R;
        nlogL = nlogL + log(Xk) + Yk'*inv(Xk)*Yk;
        end
end
```

In code listing 3.1, the function qlogL takes the matrices n, x, z, A, P, H, R and Q as parameters, and returns nlogL, the log likelihood of Q, as an output. However, MATLAB's fminsearch function returns a scalar, and the output nlogL from code listing 3.1 above happens to be a matrix.

Before employing fminsearch, it was necessary to compute a scalar measure of the output matrix nlogL that could mimic the magnitude of the matrix. The matrix determinant serves this purpose well enough, and so a line was added at the end of function qlogL to compute the determinant of nlogL, and code listing 3.1 was modified as follows to return the determinant instead as the output:

Code Listing 3.2: Modified MATLAB function for equation (3.20)

```
functionnlogL = qlogL(n,x,z,A,P,H,R,Q)
        Pk = 0;
        mlogL = 0;
        for k=1:n
        Pk_ = A*Pk*A' + Q;
        Yk = zk - H*xk_;
        Xk = H*Pk_*H' + R;
        mlogL = mlogL + log(Xk) + Yk'*inv(Xk)*Yk;
        end
        nlogL = det(mlogL);
end
```

The function qlogL can now be passed as a parameter to 'fminsearch' in MATLAB, which attempts to find the matrix **Q** for which the function qlogL returns the smallest value for nlogL.

MATLAB's 'fminsearch' function accepts a variety of parameters in different ways, depending on the task to be performed. For the purpose of the task defined above, the following variant of 'fminsearch' was employed:

Code Listing 3.3: MATLAB's fminsearch application to equation (3.20)

```
Qmin = fminsearch(@(Q) qlogL(n,x,z,A,P,H,R,Q), Q0);
```


In code listing 3.3, 'fminsearch' accepts qlogL as the function over which to carry out minimization. The @Q handle specifies which of the variables in qlogL is to be minimized, and Q0 provides the initial Q with which 'fminsearch' begins to search for the minimum Q.

3.3 Testing

For testing purposes, a section of the filter model was designed to measure percentage rms (root mean square) error and filter efficiency.

Percentage Root Mean Square (RMS) Error

The percentage rms error was calculated as follows:

First, the percentage error was calculated:

$$\%\ error = \frac{actual\ traffic - predicted\ traffic}{actual\ traffic} \quad ----- \quad (3.21)$$

Then the root mean square value of this error was computed. With LabVIEW, this is very much simplified with the aid of the RMS virtual instrument (vi) found in the LabVIEW statistics toolbox.

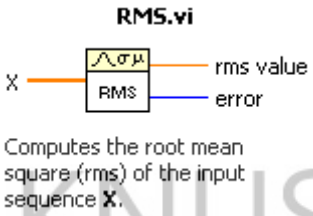


Fig 3.3: LabVIEW RMS block

The output of equation (3.21) was fed into this RMS block to compute the rms value. This rms value was then multiplied by 100 to convert it to percentage value.

Filter efficiency measurement

Efficiency was measured by the following approach:

- An acceptable rms percentage error value, say rms_acceptable, is entered into the filter by means of an input box provided on the front panel of the LabVIEW model.
  rms_acceptable represents the amount of prediction error that the user is prepared to accept when using the filter.
- On each iteration of the filter, the percentage rms error, say rms_current, is calculated
- rms_current is compared with rms_acceptable.
    - If rms_current is less than or equal to rms_acceptable, then the filter has produced a satisfactory result, and the current prediction is counted as a successful prediction.
    - Otherwise, the current prediction is counted as a failure.

- Over progressive filter iterations, the filter efficiency is cumulatively calculated as the ratio of the number of successful predictions to the total number of predictions.

Thus, for this filter, the efficiency was measured based on the acceptable percentage rms error entered.

While the filter is run, the mean square error between the predictions and the actual values is monitored. The final efficiency is the efficiency measured when the mean square error (mse) stabilizes in value.

Test Data

To achieve good network traffic prediction, different filter parameters must be increased or decreased respectively for traffic with different characteristics.

In order to investigate relationships between these filter parameters and filter efficiency for different traffic characteristics, the filter was first run on three sets of **experimental data** possessing different characteristics. The actual experimental data used are provided in Appendix 4.1. Table 3.2 shows the characteristics of the experimental data used:

Table 3.2: Characteristics for Experimental Data Sets

Data Set	Autocorrelation (Rxx)	Standard deviation (σ)
1	0.0112551	0.28
2	1	0.71
3	4.1209	1.14

The effect of varying different filter parameters was observed for each of the experimental data sets.

From these first sets of tests, a general sense is gained of how different filter parameters affect filter efficiency on different types of traffic data. With this knowledge, suitable filter parameter values can be chosen for different types of network traffic data, in order to maximize the filter efficiency.

Thus, a second set of tests was run on three sets of actual **network traffic data** possessing different characteristics. Parameters were chosen according to knowledge gained from the first set of tests with experimental data, in order to maximize the prediction efficiency. The actual experimental data used are provided in Appendix 4.2. Table 3.3 shows the characteristics of the network data used:

Table 3.3: Characteristics for Network Data Sets

Data Set	Autocorrelation (Rxx)	Standard deviation (σ)
1	14175.3	55.81
2	125713	100.74
3	450832	97.08

Filter Initialization

Before the filter could begin its iterative prediction process, it was necessary to initialize a number of variables:

- n (history length, i.e. the number of previous traffic measurements for the filter to 'remember')
- H (output gain)
- R (measurement noise covariance value)
- Q (process noise covariance)
- P (a priori covariance)

Optionally, the time delay (in milliseconds) between successive filter iterations may be set. This delay is necessary to make the changes in the actual and predicted traffic values observable. The default delay is 500 ms.

Parameter variation

The LabVIEW model developed provides options in the front panel for various filter parameters to be varied. These parameters include (descriptions as above):

- n
- R
- Initial Q value Q_0
- Initial P value P_0

Of interest is the filter efficiency at different values of the above-listed variables.

To investigate the effects of history length on the filter performance, the filter was run with different values of n:

- n = 2 (small value)
- n = 5 (normal value)
- n = 10 (large value)

For each of the above values of n, the filter efficiency was evaluated for small and large values of P_0, Q_0 and R respectively. It was of interest to observe the independent effect of varying any one of the variables P_0, Q_0or R. Hence, while each of these parameters was varied, the others were held constant.

Thus, there is no comparison between efficiency values obtained when varying P_0 and those obtained when varying Q_0 or R. Useful information is obtained when comparing efficiencies for different values of the same variable, other variables remaining constant.

With this in mind, the following table format was developed to record and observe the filter efficiency, using the experimental data sets:

Table 3.4: Table format for filter efficiency testing using experimental data sets

	P_0 = small P_0 = large	Q_0 = small Q_0 = large	R = small R = large
n = small n = medium n = large			

This table structure, populated with the respective efficiency values, gives a lucid representation of how the filter performance varies with varying P_0, Q_0, R, or n.

After compiling filter efficiencies for the different data types, trends can be observed relating filter efficiency to different values of P_0, Q_0, R and n. From the trend information observed, a good idea is obtained of how to vary these parameters, for any particular type of network traffic data, in order to maximize the prediction efficiency.

Table 3.5 shows the table format for recording the observed filter efficiencies when the filter was run for the actual network traffic data, after varying the filter parameters to maximize prediction efficiency:

Table 3.5: Table format for filter testing using actual network data sets

Network Traffic Data	Rxx	σ	n	P_0	Q_0	R	Filter Efficiency (%)
Data 1							
Data 2							
Data 3							

4. RESULTS AND ANALYSIS

In order to understand the results obtained from the tests, it would be useful to understand the different filter parameters and their implications for the filter.

i) The history length (n) represents the number of previous data values. The filter attempts to utilize any trend information present in the data history in order to predict the next value. For random data, different values of n should have little or no effect on filter efficiency, because the data history contains no meaningful trend information.

ii) The a-priori covariance (P) is a measure of the prediction error, and takes the process noise covariance (Q) into consideration. The filter usually arrives very quickly at a stable value for P, regardless of the initial value P_0. As such, the value P_0 generally should have little or no effect on the filter efficiency.

iii) The process noise covariance (Q) represents the noise introduced into the prediction process as a result of imperfections in the prediction model developed. Thus, for example, a low value for Q implies a good model and causes the filter to place more confidence in the a-priori prediction than in the actual measured value. Q_0 is the initial estimate for Q.

iv) The measurement noise covariance (R) represents the noise introduced into the prediction process as a result of errors in measurement of the actual values. Thus, for example, higher values of R imply unreliable measurements and cause the filter to place less weight on the actual data measurements and more weight on the a-priori predictions.

With this understanding of the various filter parameters, the observations from the test results can be better interpreted.

4.1 Experimental data results

Tables 6.1, 6.2 and 6.3 in Appendix 5 show the results of running the filter on data sets 1, 2 and 3.

For the different types of experimental data, a number of general observations emerged relating to the different filter parameters:

a) Weak autocorrelation (data set 1):

i. In general, different values of n had little effect on the filter efficiency, except for very large values of n where higher efficiencies were observed under certain conditions.

ii. Efficiency remained relatively unchanged for different values of P_0.

iii. Except for small values of n, efficiency increased for larger values of Q_0. For small values of n, efficiency decreased slightly with increasing Q_0.

iv. Efficiency generally increased for smaller values of R.

Figs 4.1 (a), (b) and (c) show a graphical representation of the above highlights:

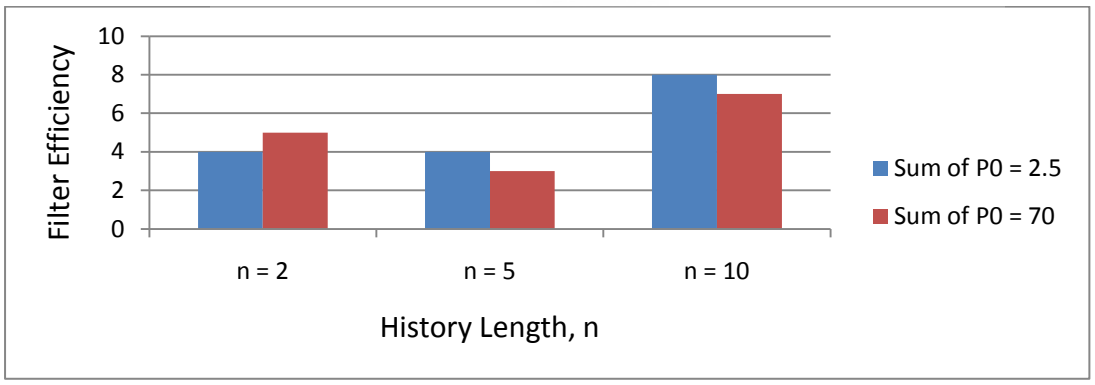


Fig 4.1(a): Effect of n and P_0 on filter efficiency for data set 1 (weak autocorrelation)

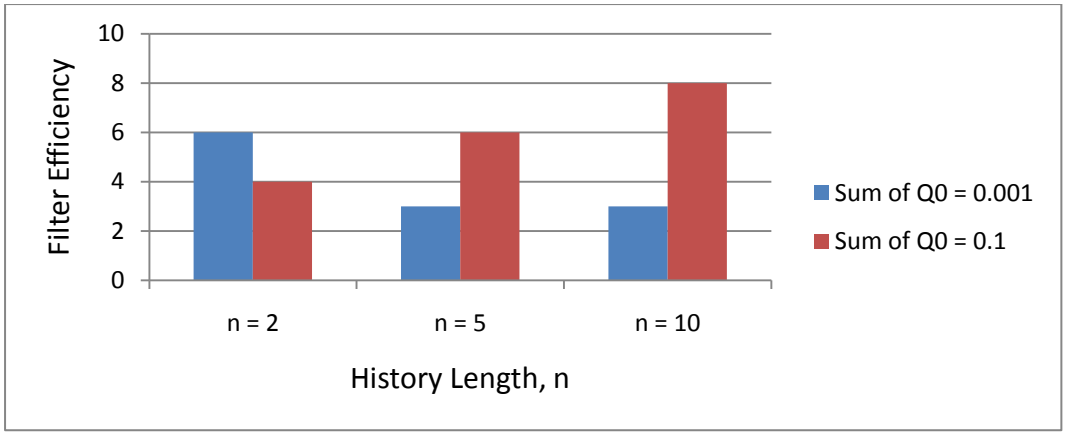


Fig 4.1(b): Effect of n and Q_0 on filter efficiency for data set 1 (weak autocorrelation)

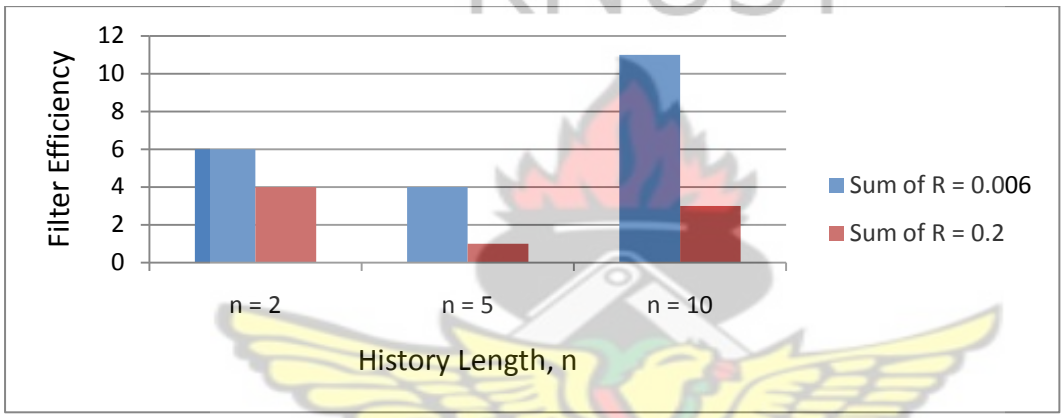


Fig 4.1(c): Effect of n and R on filter efficiency for data set 1 (weak autocorrelation)

b) Moderate autocorrelation (data set 2):

i. Except for very low values of n, efficiency was relatively unaffected by different values of n. For very low values of n, efficiency was reduced.

ii. Efficiency remained relatively unchanged for different values of P_0.

iii. Efficiency remained relatively unaffected for different values of Q_0, except for moderate values of n, where a larger Q_0 resulted in higher efficiency.

iv. Efficiency remained relatively unaffected for different values of R, except for small values of n, where a larger R resulted in higher efficiency.

Figs 4.2 (a), (b) and (c) show a graphical representation of the above highlights:

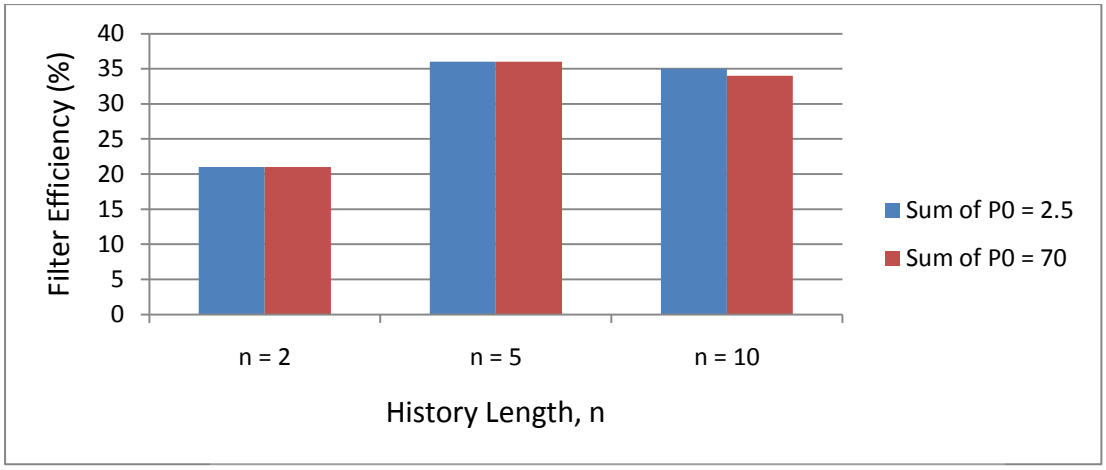


Fig 4.2(a): Effect of n and P_0 on filter efficiency for data set 2 (moderate autocorrelation)

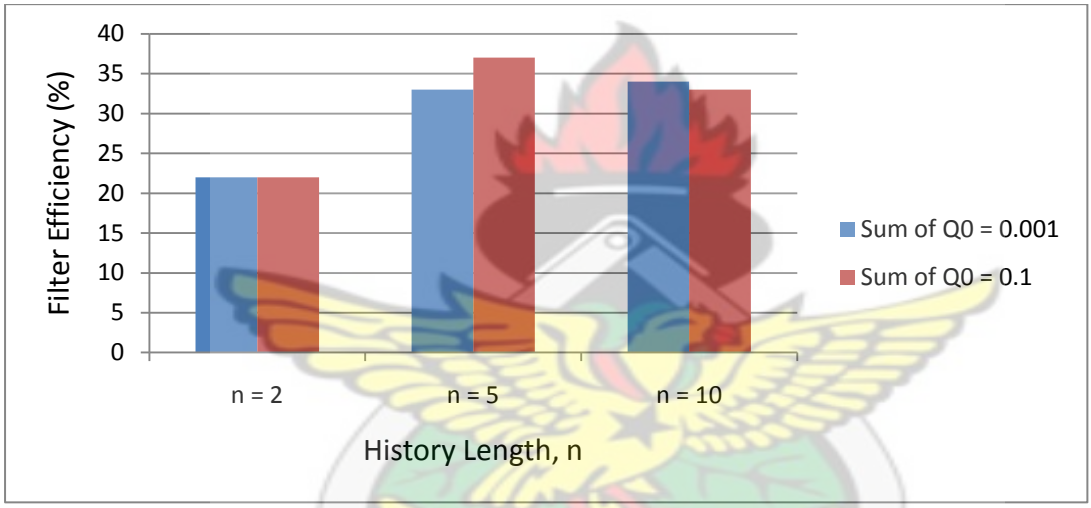


Fig 4.2(b): Effect of n and Q_0 on filter efficiency for data set 2 (moderate autocorrelation)

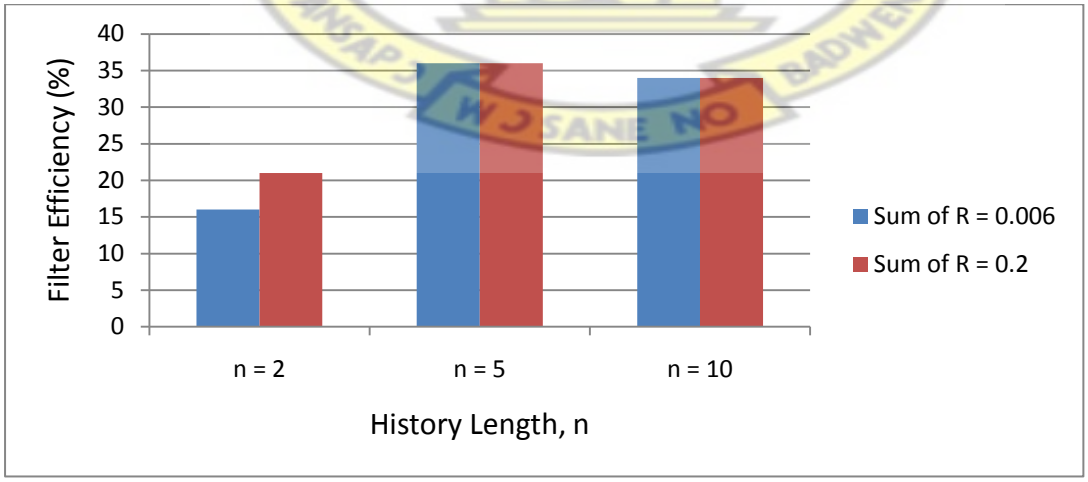


Fig 4.2(c): Effect of n and R on filter efficiency for data set 2 (moderate autocorrelation)

c) Strong autocorrelation (data set 3):

i. In general, the larger the value of n, the higher the efficiency.

ii. Efficiency remained relatively unaffected for different values of P_0, except for small values of n, where a smaller P_0 resulted in *slightly* higher efficiency.

iii. Efficiency remained relatively unaffected for different values of Q_0, except for moderate values of n, where a larger Q_0 resulted in *slightly* higher efficiency.

iv. Efficiency remained relatively unaffected for different values of R, except for moderate values of n, where a smaller R resulted in *slightly* higher efficiency.

Figs 4.3 (a), (b) and (c) show a graphical representation of the above highlights:

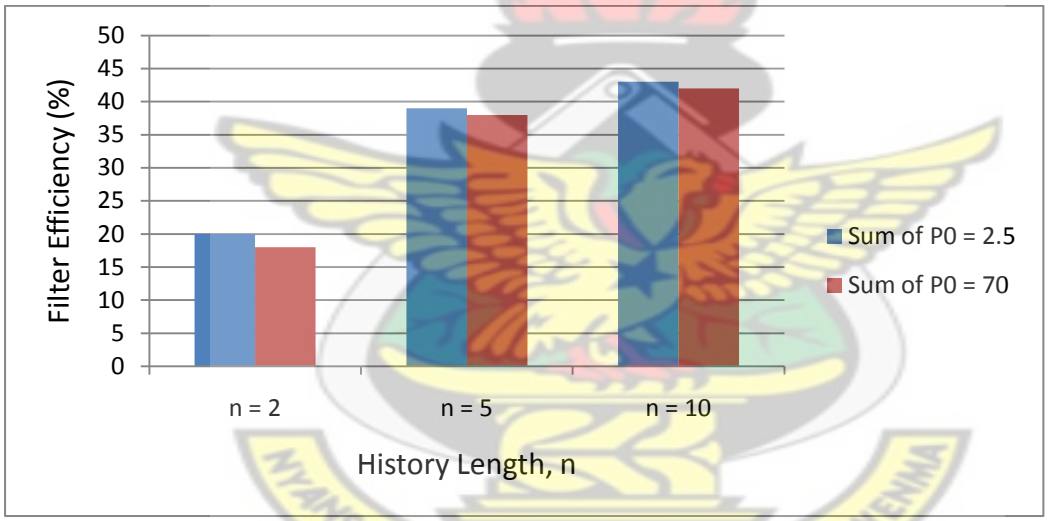


Fig 4.3(a): Effect of n and P_0 on filter efficiency for data set 3 (strong autocorrelation)

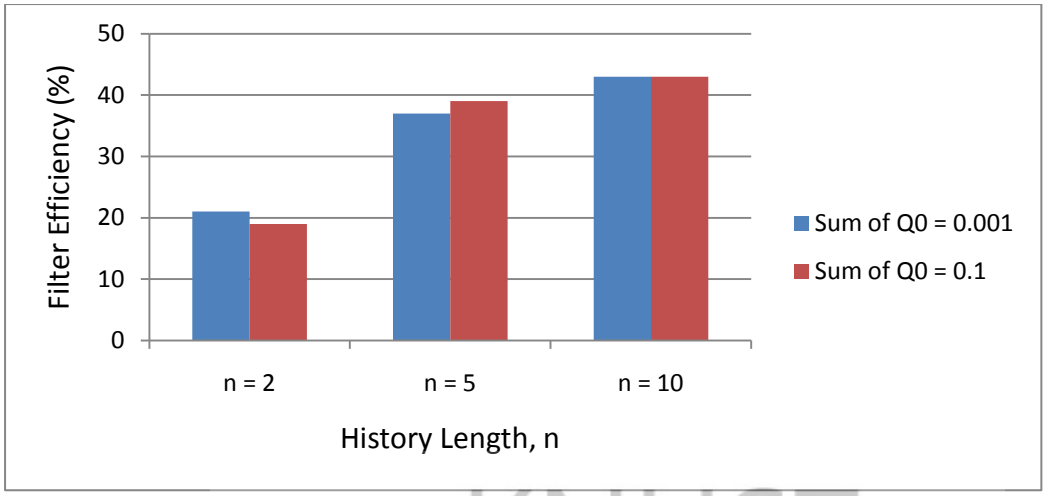


Fig 4.3(b): Effect of n and Q_0 on filter efficiency for data set 3 (strong autocorrelation)

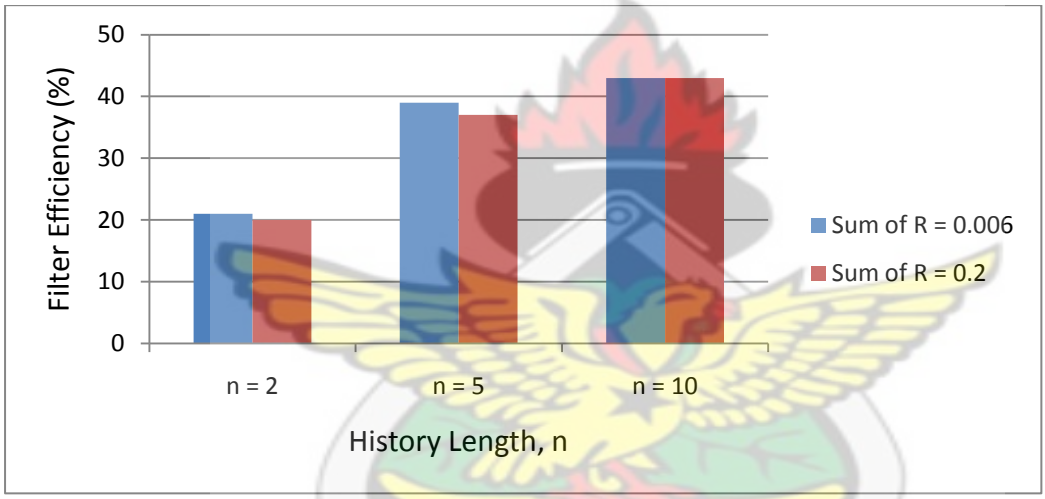


Fig 4.3(c): Effect of n and R on filter efficiency for data set 3 (strong autocorrelation)

Analysis of experimental data results

Effect of history length n:

Previous data values can give information about trends inherent in the data. For data with weak autocorrelation (simulating random data), previous data values hold little or no trend information. The stronger the autocorrelation, the more trend information is available from previous data values. Thus, the filter was able to achieve better prediction accuracy with a

larger value for n with a data set that has stronger autocorrelation. This was confirmed by the results described above.

Effect of P_0:

As already discussed above, the filter quickly attained a self-stabilizing value for P regardless of the initial value P_0. Therefore different values of P_0 should have no effect on the filter efficiency. This was readily verified from the general trend in the results above.

Effect of Q_0:

For data set 1 (weak autocorrelation), larger values of Q_0 generally produced higher efficiencies. This was due to the fact that previous data values for this data set contained no meaningful trend information. Hence, whatever 'trend' the filter extracted from previous data was actually a false trend. Larger Q values informed the filter to place less weight on the prediction process and more weight on the actual measurement values. This offset the effects of false trends extracted from the previous data values.

The exception was with small values of n, where efficiency *increased* slightly with smaller Q_0. This is because the 'no trend' information extracted by the filter from smaller values of n actually turned out to be accurate. In this case, a smaller Q informed the filter to place more weight on the prediction process.

For data sets 2 and 3 with better autocorrelations, different values of Q_0generally had little effect on the filter efficiency. This is because whether the filter placed more or less weight on the prediction process or not, the actual measurements were correlated enough to provide useful trend information for the filter.

Effect of R:

R, representing the measurement noise covariance, informs the filter about how much weight to place on the actual measurement values. Smaller values of R imply more reliable measurements, and vice-versa.

For data set 1, which simulated random data, little weight could be placed on the actual prediction process itself. Therefore, better efficiencies were obtained by placing more weight on the actual measurements. This is why smaller values of R resulted in higher efficiencies for data set 1.

For data sets 2 and 3, with good autocorrelation, R *generally* had little effect on filter efficiency. This is because the trend information obtained from the previous data values was good enough to overshadow the weight (or lack of weight) placed on the actual measurement values.

However, for small values of n on data set 2 (moderate autocorrelation), a larger R produced higher efficiency. This may be attributed to the fact that data sets with good autocorrelation are usually stationary. This means that small values of n may contain more trend information than actual single data measurements. Thus, higher efficiencies can be obtained by placing less weight on the actual measurements, which is achieved by setting R to a larger value.

Data set 3 (strong autocorrelation) may also be treated as a stationary process. Thus, a collection of data values from such a set holds good trend information. For a data set with greater autocorrelation, small or large collections of data values hold good trend information. However, for a moderately-sized collection of data values, the trend displayed

amongst the individual measurements may be a more accurate description than that displayed by the measurements taken as a group. In this case better prediction efficiency would be achieved by placing more weight on the actual measurement values. This is achieved by setting R to a smaller value. The slightly higher efficiency recorded for a smaller R value when n is of moderate size for such a data set could be an occurrence of this phenomenon.

Guidelines for network traffic data prediction:

Having obtained the above information regarding the relationships between different filter parameters and filter efficiency, a reasonable set of guidelines may be drawn up for achieving better prediction efficiency with network traffic data. As previously stated, network traffic data is known to exhibit strong autocorrelation [10]. Therefore the parameter values that produced higher efficiencies for the experimental data sets with good autocorrelation serve as a good guide to choosing parameter values when testing with actual network traffic data.

Based on the above results, the following criteria may be set up. For network traffic data (very strong autocorrelation), better prediction efficiency can be achieved with:

a) Larger values of n (not too large, as trend information may be distorted if n is too large)
b) For moderate values of n:
    i) Larger values of Q_0.
    ii) Smaller values of R. For network traffic data, R is usually fixed because it is a representation of errors in measurement arising from imperfections in the

instruments used to measure the traffic (measurement error is usually fixed for a particular measuring instrument). It is therefore important in this case to choose traffic data measured with instruments with less error, i.e. more accurate instruments.

(The value of P_0 has been shown to have little or no effect on the filter efficiency).

With this set of guidelines, a clearer idea was obtained on how to vary the filter parameters to achieve good filter efficiency when running the filter with actual network traffic data.

4.2 Network data results

The filter was finally tested with actual network traffic data. Using the guidelines just listed above, the filter parameters were varied and adjusted to achieve the maximum efficiency attainable for the different types of network traffic data used.

Table 6.4 of Appendix 5 (reproduced here for convenience) shows the results obtained from this process:

Table 6.4: Table of filter efficiencies for network traffic data

Network Traffic Data	Rxx	σ	n	P_0	Q_0	R	Filter Efficiency (%)
Data 1	*14175.3*	55.81	10	*2.5*	*0.25*	*0.0006*	*85*
Data 2	*125713*	100.74	15	*2.5*	*0.025*	*0.015*	*86*
Data 3	*450832*	97.08	13	*2.5*	*0.0185553*	*0.015*	*88*

The efficiencies shown in Table 6.4 were obtained after varying n, Q_0 and R according to the guidelines above. It is worth noting the following:

i) The values shown for Q_0 and R are not unique values for which the maximum efficiencies are obtained.

ii) The final value for n in Table 6.4 is the smallest value of n for which the maximum efficiency, quoted in the table, is achieved.

Fig 4.4 is a graph of the efficiencies obtained for the different data sets after setting n, Q_0 and R as shown in Table 6.4:

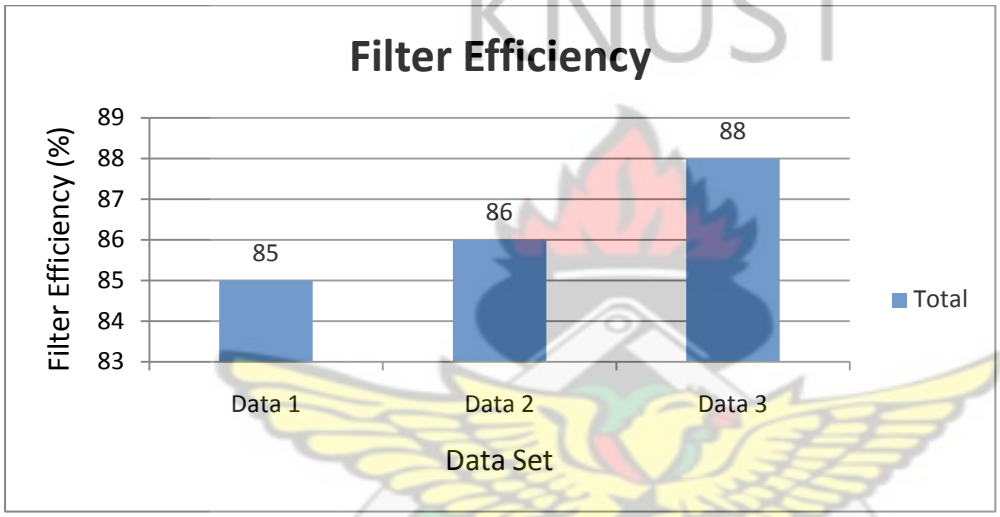


Fig 4.4: Filter efficiencies for the different network data samples

Analysis of network data results

The filter parameters that produced the maximum efficiencies quoted in the Table 6.4 were of such characteristics as to agree with the criteria obtained from the tests with the experimental data.

History length n:

For these very highly autocorrelated data sets, it was found that larger values of n produced higher efficiencies.

For example, for data 1, higher efficiencies were obtained as n was increased up to n = 10. Above n = 10, the efficiency did not show any notable improvement. For data 2 and data 3, efficiency kept increasing up to values of n = 15 and n = 13 respectively.

This disparity in minimum n for which maximum efficiency is reached can be explained by the different standard deviation values of the three different data sets. For data 1, with the smallest standard deviation of 55.81, the filter reached maximum efficiency at a smaller n value of 10. The highest n value of 15 was required for data 2, the data set with the highest standard deviation of 100.74. Thus, it becomes obvious that data with a smaller standard deviation and a bigger autocorrelation lends itself to better prediction efficiency with this filter.

Q_0 and R:

Because higher values of n were used for these particular data sets, the values of Q_0 and R did not exert much influence on the efficiency. This is because according to the results obtained with the experimental data sets, Q_0 and R have a notable effect when moderate values of n are used. Hence, for these particular data sets, the values shown in the table for Q_0 and R were not the only values for which the maximum efficiencies are obtained. In other words, the Q_0 and R values shown in this table are not unique values for maximum efficiency in this particular test.

P_0:

The P_0 values were the same in the table because the results were the same no matter value P_0 was set to. This is consistent with the fact that the filter always reaches an optimum value for P (a-priori covariance) no matter what the initial value P_0 is.

5. SUMMARY

5.1 Goal Attainment

In this project, a Kalman filter model was successfully developed to predict network traffic. Traffic behaviour was successfully captured in the model, and the filter was able to predict network traffic to an appreciable degree of accuracy, approaching 90%.

Admittedly the model is not perfect. Much more work is needed to fine-tune the behaviour of the model, enabling it to carry out its predictions with a much higher degree of accuracy, with a smaller error tolerance.

The filter was able to handle data of varying characteristics, but it is best-suited for data that exhibits strong autocorrelation and low variance.

It is worth noting that this filter performs best in applications of short-term traffic forecasting.

5.2 Project Challenges

The network traffic data used in the tests consisted of traffic measurements taken at hourly intervals, as most network traffic measurements are taken at least hourly. This filter, however, was designed for short-term traffic prediction. The difficulty in obtaining traffic measurements for shorter time intervals was a major challenge in the testing of this model.

Secondly, the matrices involved in the filter's computations for history length n included some of dimension n by n. For large n values, computations can become highly memory-consuming and slow. These computational difficulties associated with dealing with large

values of n precluded the investigation of the behaviour of the filter for much larger values of n.

Nevertheless, it can be expected that for data with very strong autocorrelation, efficiency will continue to increase for slightly larger values of n. For data without very strong correlation, however, it can be extrapolated that the filter efficiency would begin to decrease at comparatively larger values of n. This is because with larger collections of previous data values, the trend information begins to blur out, causing the filter to build a misleading picture of the nature of the data.

5.3 Future research

The first point of interest for future research concerning this project would be to improve upon the prediction efficiency. Many forecasting techniques exist today that, used in conjunction with a Kalman filter, could remarkably improve the prediction efficiency of the filter. It would be worth devoting time and effort into such research.

With very good prediction efficiency, the adaptation of the filter to perform multi-step prediction becomes a viable and appealing prospect. If the prediction for time-step k+1 is always reasonably close to the true value, then it may be taken as the true value. This pseudo-true value may be used to predict the next value (i.e. for time-step k+2), even before the actual value for time-step k+1 becomes available. With the right analysis and implementation, m-step prediction becomes possible (where m is an integer).

5.4 Conclusion

The Kalman filter is a very powerful estimation and prediction tool. With the right adaptation, this filter model provides a simple, convenient and inexpensive resource for

prediction and estimation for applications in computer, telecommunications and transportation networks, to name just a few. Such applications can go a long way in providing affordable and simple technology to help in alleviating traffic congestion. Many other applications for Kalman filters, such as tracking and smoothing, can be adapted to solve, in a very cost-effective manner, many problems faced by developing countries such as Ghana.

Considering the simplicity and cost-effectiveness of Kalman filter adaptations and applications, it is technologically and economically worth investing more into Kalman filter research and applications.

6. REFERENCES

[1] “Statistical Forecasting Methods.” Internet: http://www.statisticalforecasting.com/

[2] Yuanchang Xie, Yunlong Zhang & Zhirui Ye (2007) “Short-Term Traffic Volume Forecasting Using Kalman Filter with Discrete Wavelet Decomposition.” *Computer-Aided Civil and Infrastructure Engineering.* [Online]

[3] Declan Delaney and Tomas Ward. “A Java Tool for Exploring State Estimation using the Kalman Filter” in ISSC 2004, Belfast, June 30 - July 2, pp. 2.

[4] “Kalman Filter.” Internet: http://en.wikipedia.org/wiki/Kalman_filter

[5] P. D. Joseph. “Kalman_1Lessons_0_to_4_rev 11_05” Personal e-mails

[6] “Traffic flow.” Internet: http://en.wikipedia.org/wiki/Traffic_flow

[7] “Types of Traffic Flow.” Internet: http://www.webs1.uidaho.edu/niatt_labmanual/Chapters/trafficflowtheory/theoryandconcepts/TypesOfTrafficFlow.htm

[8] “Traffic Flow Theory.” Internet: http://www.webs1.uidaho.edu/niatt_labmanual/Chapters/trafficflowtheory/professionalpractice/TrafficFlowParameters.htm

[9] Tom Lane, TheMathworks, Inc., Personal e-mails

[10] “autocorrelation of network traffic” Internet: http://www.soi.wide.ad.jp/class/20070044/slides/16/index_21.html

[11] “Short-term traffic safety forecasting using Gaussian mixture model and Kalman filter” Internet: http://link.springer.com/article/10.1631%2Fjzus.A1200218

[12] “Kalman filter approach to traffic modelling and prediction” Internet: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=932091

[13] “Tracking and predicting a network traffic process” Internet: http://www.sciencedirect.com/science/article/pii/S0169207096007005

[14] “An Extended Kalman Filter Application for Traffic State Estimation Using CTM with Implicit Mode Switching and Dynamic Parameters” Internet: https://www.mech.kuleuven.be/cib/verkeer/dwn/pub/P2007B.pdf

[15] “Real-time freeway traffic state estimation based on extended Kalman filter: a general approach” Internet: http://www.sciencedirect.com/science/article/pii/S0191261504000438

[16] “Dynamic prediction of traffic volume through Kalman filtering theory” Internet: http://econpapers.repec.org/article/eeetransb/v_3a18_3ay_3a1984_3ai_3a1_3ap_3a1-11.htm

[17] “Theory and Application of Advanced Traffic Forecast Methods” Internet: http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-5656/Chrobokdiss.pdf

[18] Moshe Ben-Akiva, Michel Bierlaire, Haris Koutsopoulos and Rabi Mishalani U. “DynaMIT: a simulation-based system for traffic prediction” in paper presented at the DACCORD Short Term Forecasting Workshop February, 1998 Delft, The Netherlands

[19] Ashok, K. and M. Ben-Akiva (1993). “Dynamic O-D Matrix Estimation and Prediction for Real Time Traffic Management Systems”. In: Transportation and Traffic Theory (Daganzo, C.F., (Ed.)). Elsevier Science Publishing Company Inc.

[20] Sage, A.P. and Husa, G.W. (1969) “Adaptive filtering with unknown prior statistics”. In: Proceedings of the Joint Automatic Control Conference, pp. 760-769

[21] “The Bayesian Approach to Forecasting” Internet: http://www.oracle.com/us/products/applications/057028.pdf

[22] “Cell Transmission Model” Internet: http://en.wikipedia.org/wiki/Cell_Transmission_Model

7. APPENDICES

APPENDIX 1: JAVA CODE

```
package Kalman;

import java.io.*;

importjava.util.*;

public class KalmanFilter

{

//          VARIABLE DECLARATION

            private static intm,n = 0;        //m = number of links, n = number of measurements

            staticinttotalIterations = 0;

//          private static String errorMsg = "";

            private static booleanprintHorizontal = false;

            private static BufferedWriter file;

            private static BufferedReaderinputData;

            private static StringTokenizerst;

            static Matrix predictedVol;    //matrix holding predicted traffic flow rate per iteration (m by 1)

            private static float predictedTraffic[][];  //matrix holding predicted traffic flow rates for all iterations (n by m)

            private static float inputTraffic[][];        //(n by m)

            static Matrix priorState;         //a priori state estimate (n by 1)

            static Matrix priorCov;           //a priori covariance (n by n)

            static Matrix postState;          //a posteriori state estimate (n by 1)

            static Matrix postCov;            //a posteriori covariance (n by n)

            static Matrix z;        //actual traffic flow rate (m by 1)

            static Matrix I;        //unit matrix (n by n)

            static Matrix A;        //state gain (n by n)

            static Matrix K;        //Kalman gain (n by m)

            static Matrix R;        //measurement noise covariance (m by m)

            static Matrix Q;        //process noise covariance (n by n)

            static Matrix H;        //output gain (m by n)
```

```
        statichistoryQueuetrafficHistory;       //queue holding last n measurements (m by n)
        statichistoryQueuestateHistory;         //queue holding last n states (n by 1)
//      static float[][] testMatrix = {{2,3},{4,5}};
//      static Matrix test;
        //constructor
        publicKalmanFilter(String dataFile, inthistoryLength) throws Exception
        {
                float[][] testData = getArray(dataFile);
                KalmanFilterkf = new KalmanFilter(testData, testData.length, historyLength);
        }
        //constructor
        publicKalmanFilter(float[][] trafficData, intnumOfIterations, inthistoryLength) throws Exception
        {
                inputTraffic = trafficData;
//              test = new Matrix(testMatrix);
//              System.out.println("test determinant = "+test.determinant());
//              System.out.println("test inverse = ");
//              test.inverse().printMatrix();
        // VARIABLE INITIALISATION
                //initialize matrices and other variables
                m = trafficData[0].length; //number of links
                n = historyLength;  //number of measurements
                totalIterations = numOfIterations;
                predictedTraffic = new float[numOfIterations][m];
                priorState = new Matrix(n,1,1.0f/n);
                priorCov = new Matrix(n,0.009f);
                postState = new Matrix(n,1,0);
                postCov = new Matrix(n,n,0);
                z = new Matrix(m,1,0);
                I = new Matrix(n,1);
                A = new Matrix(n,1);
```

```
            K = new Matrix(n,m,0);
            R = new Matrix(m,m,0.015f);
            Q = new Matrix(n,n,0.0f);
            H = new Matrix(m,n,3.0f);
            trafficHistory = new historyQueue(m,n,3.0f);
            stateHistory = new historyQueue(n,1,3.0f);
//          H = trafficHistory.update(trafficData);
            predictedVol = new Matrix(m,1,0);
            kalmanFilter(trafficData,numOfIterations);
}
public static float[][] getArray(String fileName) throws IOException
{
            inputData = new BufferedReader(new FileReader(new File(fileName)));
            String line = inputData.readLine();
            totalIterations++;
            st = new StringTokenizer(line);
            m = st.countTokens();
            line = inputData.readLine();
            while(line!=null)
            {
                        totalIterations++;
                        line = inputData.readLine();
            }
            float[][] theArray = new float[totalIterations][m];
            inputData.close();
            populateArrayFromFile(theArray,fileName);
            returntheArray;
}
public static void populateArrayFromFile(float[][] array, String file) throws IOException
{
            inputData = new BufferedReader(new FileReader(new File(file)));
```

```
                String nextLine = "";
                for(int i = 0; i<array.length; i++)
                {
                        nextLine = inputData.readLine();
                        st = new StringTokenizer(nextLine);
                        for(int j = 0; j<array[0].length; j++)
                        {
                                array[i][j] = Float.parseFloat(st.nextToken());
                        }
                }
        }
        public static void kalmanFilter(float[][] traf, int iterations) throws Exception
        {
                if(iterations==0)
                        return;
                else
                {
                        try
                        {
                                kalmanFilter(traf,iterations-1);
                                System.out.println("Iteration "+iterations+" of "+totalIterations+" in progress...");
                        //      H.printMatrix();
                                z.set1DArray(traf[iterations-1]);
                                //KALMAN FILTER EQUATIONS
                                //Predictor stage
                                priorState = A.times(priorState);          //a priori state estimate
                                priorCov = ((A.times(priorCov)).times(A.trans())).plus(Q);    //a priori cov
                                //Corrector stage
                                K =
(priorCov.times(H.trans())).times((((H.times(priorCov)).times(H.trans())).plus(R)).inverse());           //Kalman filter gain
                                postState = priorState.plus(K.times(z.minus(H.times(priorState))));    //a posteriori
state estimate
```

```
                                        postCov =
(((I.minus(K.times(H))).times(priorCov)).times((I.minus(K.times(H))).trans())).plus((K.times(R)).times(K.trans()));          //a
posteriori covariance

                                        //predicted output traffic flow rate

                                        H = trafficHistory.update(traf);

                                        predictedVol = H.times(postState);

                                        predictedTraffic[iterations-1] = predictedVol.get1DArray();

                                        //update a priori state and covariance

                                        priorState.setMatrix(postState);

                                        priorCov.setMatrix(postCov);

                                        System.out.println("Iteration "+iterations+" completed.\n");

                                        return;

                                }

                                catch(Exception e)

                                {

                                        System.out.println(e);

                                //      errorMsg = e.toString();

                                //      System.exit(1);

                                }

                        }

                }

                public void print(float[][] matrix) throws IOException

                {

                        intnumOfLinks = matrix[0].length;

                        file = new BufferedWriter(new FileWriter(new File("Kalman Test Results for test data.txt")));

                        if(printHorizontal)

                        {

                                System.out.println("\t\t\tPredicted Traffic Data\n");

                                file.write("\t\t\tPredicted Traffic Data");

                                file.newLine();

                                file.newLine();

                        //      System.out.print("Link");
```

```
		for(int i = 1; i<=numOfLinks; i++)
		{
			System.out.print("\tLink "+i+"\t\t");
			file.write("\tLink "+i+"\t\t");
		}
		System.out.println("\n");
		file.newLine();
		file.newLine();
		for(int i = 1; i<=numOfLinks; i++)
		{
			System.out.print("\tPredicted\tActual");
			file.write("\tPredicted\tActual");
		}
		System.out.println("\n");
		file.newLine();
		file.newLine();
		for(int i = 0; i<matrix.length; i++)
		{
			System.out.print(i+1);
			file.write((i+1)+"");
			for(int j = 0; j<numOfLinks; j++)
			{
				System.out.print("\t"+matrix[i][j]+"\t"+inputTraffic[i][j]);
				file.write("\t"+matrix[i][j]+"\t"+inputTraffic[i][j]);
			}
			System.out.println();
			file.newLine();
		}
		System.out.println("\n\n");
}
else
```

```
		{
			System.out.println("\t\t\tPredicted Traffic Data\n");
			file.write("\t\t\tPredicted Traffic Data");
			file.newLine();
			file.newLine();
			for(int i = 0; i<numOfLinks; i++)
			{
				System.out.println("Link "+(i+1)+"\t");
				file.write("Link "+(i+1)+"\t");
				file.newLine();
				System.out.println("\tPredicted\tActual");
				file.write("\tPredicted\tActual");
				file.newLine();
				for(int j = 0; j<matrix.length; j++)
				{
					System.out.println(j+1+"\t"+matrix[j][i]+"\t"+inputTraffic[j][i]);
					file.write(j+1+"\t"+matrix[j][i]+"\t"+inputTraffic[j][i]);
					file.newLine();
				}
				System.out.println();
				file.newLine();
			}
			System.out.println("\n\n");
		}
		file.close();
	}
	//main method
	public static void main(String[] args) throws Exception
	{
//		/*
			printHorizontal = true;
```

```
float[][] testData =  {// link 1 link 2 link 3
                        {3.0f,  9.5f,  2.2f},    //row 1
                        {5.1f,  8.9f,  3.1f},    //row 2
                        {4.8f,  9.3f,  5.2f},    //row 3
                        {6.4f,  10.1f, 9.0f},    //row 4
                        {8.2f,  11.5f, 6.8f},    //row 5
                        {6.3f,  11.1f, 6.1f},    //row 6
                        {5.5f,  12.3f, 5.5f},    //row 7
                        {4.8f,  11.3f, 5.5f},    //row 8
                        {5.6f,  10.4f, 3.8f},    //row 9
                        {6.3f,  8.9f,  5.1f},    //row 10
                        {9.2f,  10.1f, 6.8f},    //row 11
                        {8.5f,  12.3f, 5.2f},    //row 12
                        {6.8f,  10.3f, 5.5f},    //row 13
                        {5.6f,  9.4f, 4.8f},                //row 14
                        {6.3f,  8.9f,  3.1f}                //row 15
                        };          //10 by 3
//          */
            /*
                printHorizontal = true;
                float[][] testData =  {// link 1
                        {116.8f},  //row 1
                        {120.1f},  //row 2
                        {123.2f},  //row 3
                        {130.2f},  //row 4
                        {131.4f},  //row 5
                        {125.6f},  //row 6
                        {124.5f},  //row 7
                        {134.3f},  //row 8
                        {135.2f},  //row 9
                        {151.8f},  //row 10
```

```
                    {146.4f},  //row 11
                    {139.0f},  //row 12
                    {127.8f},  //row 13
                    {147.0f},  //row 14
                    {165.9f},  //row 15
                    {165.5f}                //row 16
                    };          //10 by 1
*/
/*
        printHorizontal = false;
        float[][] testData =  {// link 1  link 2 link 3 link 4   link 5
                    {0.2f,  1.1f,  2.8f,  22.07f,  19.12f},       //row 1
                    {2.4f,  2.3f,  8.7f,  19.88f,  14.02f},       //row 2
                    {4.6f,  0.3f,  9.4f,  18.27f,  25.09f},       //row 3
                    {6.5f,  8.8f,  5.3f,  16.09f,   2.89f},       //row 4
                    {5.3f,  7.7f,  1.9f,  14.35f,  22.12f},       //row 5
                    {3.2f,  0.4f,  2.1f,   3.03f,  11.04f},       //row 6
                    {2.9f,  5.9f,  0.5f,  20.09f,   9.03f},       //row 7
                    {9.6f,  9.7f,  0.8f,   8.02f,  15.01f},       //row 8
                    {6.8f,  4.2f,  3.0f,  19.81f,   7.08f},       //row 9
                    {0.8f,  6.9f,  2.4f,  19.57f,  22.11f},       //row 10
                    {5.0f,  3.2f,  7.2f,  52.50f,  22.15f},       //row 11
                    {4.1f,  6.1f,  3.7f,  20.00f,  13.40f},       //row 12
                    {2.5f,  1.5f,  3.1f,  20.64f,  18.39f},       //row 13
                    {2.0f,  2.2f,  3.9f,  22.28f,   7.07f}        //row 14
                    };          //14 by 5
*/
        System.out.println("\n\t\t\tKalman Filter Test\n\n");
        KalmanFilterkalmanTest = new KalmanFilter(testData,testData.length,5);
//      KalmanFilterkalmanTest = new KalmanFilter("test data 2.txt",10);
        kalmanTest.print(predictedTraffic);
```

```
		//		System.out.println(predictedTraffic);
		}
}
//HISTORY QUEUE
package Kalman;
public class historyQueue
{
		inti,j = 0;
		intcopyMarker,nextDataIndex,queueSize,currentSize,queueWidth;	//queue markers
		Matrix history;		//queue to 'remember' previous n traffic data
		publichistoryQueue(int row, int col, float value)
		{
			copyMarker = 0;
			nextDataIndex = 0;
			currentSize = 0;
			queueSize = col;
			queueWidth = row;
			history = new Matrix(row,col,value);
		}
		public Matrix update(float[][] data)
		{
			if(nextDataIndex == data.length-1)
				return history;
			//this loop to prevent array index of -1
			//copies each column of 'history' to the one directly after it
			if(currentSize!=0)
				for(copyMarker = (currentSize==queueSize)? currentSize - 2:currentSize - 1; copyMarker>=0;
copyMarker--)
					for(i = 0; i<queueWidth; i++)
						history.setValue(i,copyMarker+1,history.value(i,copyMarker));
			//this loop copies next column of data array into
```

```
                //first column of 'history'
                for(i = 0; i<queueWidth; i++)
                        history.setValue(i,0,data[nextDataIndex][i]);
                if(currentSize<queueSize)
                        currentSize++;
                nextDataIndex++;
                return history;
        }
}
```


APPENDIX 2: MATLAB CODES

2.1: LabVIEW MATLAB Script Node Code

```
if i == 0    % This 'if' block necessary because on first code iteration, P and H are 2-D, so cat(3,P,p) function call would fail
P = p;
H = h;
else
P = cat(3,P,p);
H = cat(3,H,h);
end
if i == N-1
cd ('path');          % 'path' should be the full path to the directory where qlogL is located
Q = fminsearch(@(Q) qlogL(N,x,z,A,P,H,R,Q), Q);
end
```

2.2: qlogL Code

```
functionnlogL = qlogL(n,x,z,A,P,H,R,Q)
mlogL = 0;
for k=1:n
P(:,:,n) = A*P(:,:,n)*A' + Q;
```

```
Y = z(:,n) - H(:,:,n)*x(:,n);

X = H(:,:,n)*P(:,:,n)*H(:,:,n)' + R;

mlogL = mlogL + log(X) + Y'*inv(X)*Y;

end

nlogL = det(mlogL);

end
```

APPENDIX 3: LABVIEW MODEL

a) Front panel

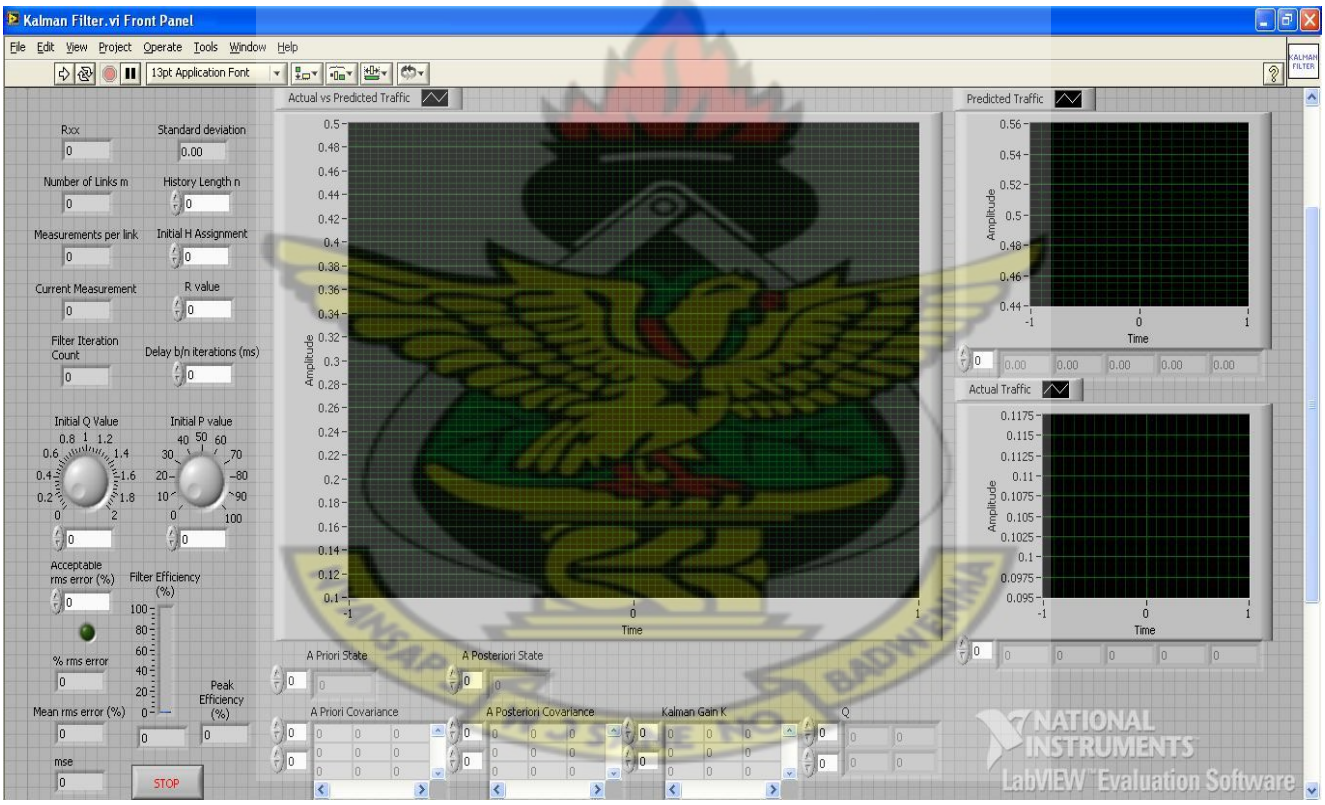


Fig. 6.1: LabVIEW front panel diagram

b) Block diagram

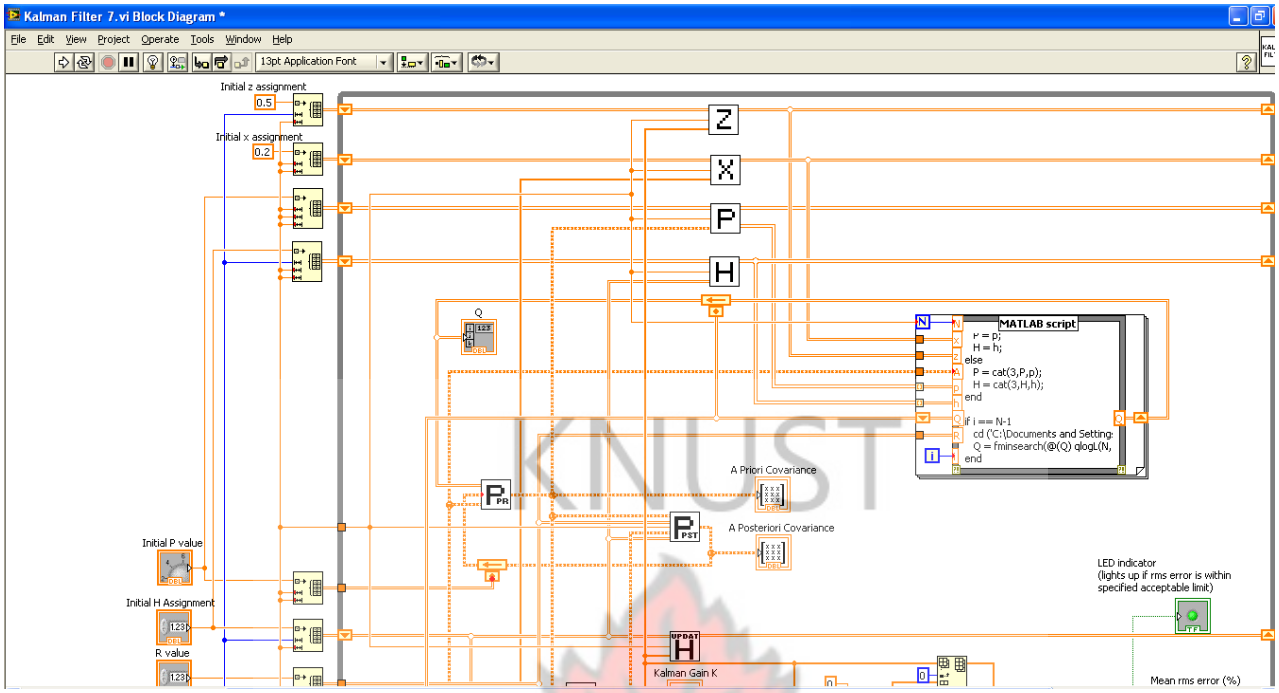


Fig. 6.2a: LabVIEW block diagram (top half)

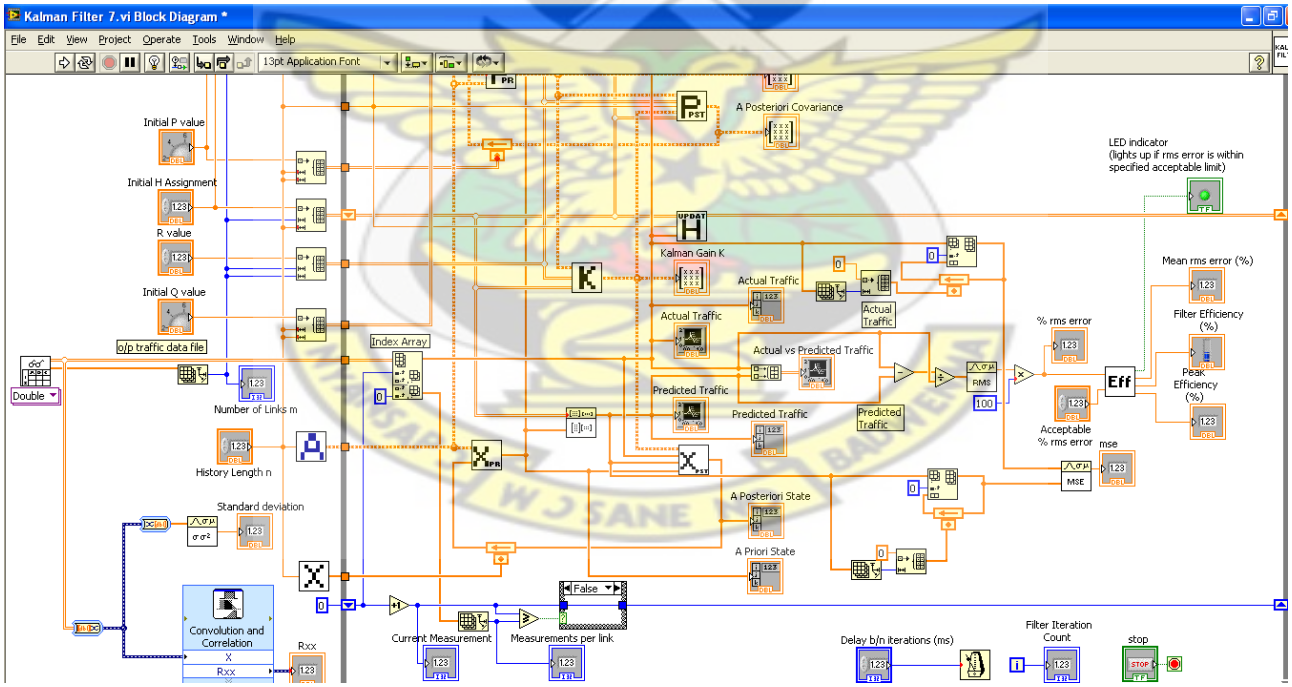


Fig. 6.2b: LabVIEW block diagram (bottom half)

APPENDIX 4: DATA SETS

4.1: Experimental Data

Data Set 1 (Rxx = 0.0112551,σ = 0.28)

0.10609	0.16852	0.23700	0.18022	0.18094	0.65983	0.65235	0.21594	0.11515	0.00067	0.00387
0.25202	0.65011	0.53646	0.08462	0.05640	0.26630	0.03641	0.26030	0.03995	0.33612	0.11672
0.69775	0.07569	0.36149	0.42341	0.26721	0.00363	0.08333	0.22480	0.13728	0.12698	0.18770
0.34741	0.54947	0.57423	0.57493	0.28188	0.58438	0.50236	0.60766	1.04784	0.68857	0.11450
0.22005	0.36533	0.00151	0.03740	0.22438	0.16196	0.12862	0.08448	0.17945	0.37502	0.95727
0.86188	0.39464	0.53794	0.13054	0.39138	1.23825	0.66286				

Data Set 2 (Rxx = 1,σ = 0.71)

1.00	0.6	1.86	0.94	1.00	0.75	0.81	0.72	0.81	1.25	2.03	2.86
1.64	0.47	0.14	6.67	3.36	2.53	3.06	4.28	6.31	8.56	10.44	10.06
12.17	12.97	14.64	18.61	17.06	20.36	17.81	16.75	23.83	24.69	32.25	30.06
30.69	32.64	35.19	36.53	35.14	39.53	49.36	47.36	50.58	0.14	0.31	0.14
0.11	0.08	0.78	1.28	1.53	0.14	0.53	0.81	0.58	1.61	0.72	0.03
0.22	0.44	0.17	0.39	0.39	1.14	1.08	1.36	1.44	1.72	1.03	1.06
1.03	0.56	0.47	0.50	0.39	0.28	0.28	0.81	0.42	1.25	0.89	1.08
0.39	0.06	0.28	0.06	0.78	0.25	2.06	1.25	0.92	1.44	1.92	2.08
2.17	2.50	3.17	3.22	2.97	4.06	1.72	2.25	2.03	3.61	2.61	1.47
0.14	0.00	0.00	0.00	0.03	0.06	0.22	0.14	0.33	0.58	0.28	0.39
0.39	0.14	0.31	1.00	0.78	1.17	0.25	0.50	1.00	0.86	0.17	0.17
0.61	1.19	0.11	0.06	0.25	1.03	1.17	1.28	2.31	2.25	1.69	1.58
1.22	2.67	2.03	1.69	1.97	2.56	2.53	2.22	2.67	0.58	0.22	3.19
0.83	2.00	2.97	4.14	5.94	9.17	14.08	16.25	20.11	22.50	22.11	18.83
18.56	17.11	17.42	20.19	24.44	34.17	43.33	41.28	29.00	17.83	6.69	0.33
0.06	0.06	0.22	0.28	0.92	2.14	2.14	2.42	2.69	2.69	2.28	2.00
2.64	3.28	2.97	2.86	3.00	4.53	4.64	3.92	3.39	1.86	0.69	

Data Set 3 (Rxx = 4.1209,σ = 1.14)

2.03	3.17	3.08	3.94	2.92	2.19	2.22	2.67	2.56	2.17	2.86	2.72
3.22	5.14	6.11	3.72	2.17	2.17	1.22	1.17	1.86	2.64	1.33	1.53
1.86	2.56	3.69	2.36	4.28	2.11	2.25	3.56	3.36	2.89	1.47	1.97
2.81	3.50	3.17	3.67	2.58	2.44	1.94	2.22	2.69	2.22	2.69	4.69
6.44	5.81	2.78	1.72	1.89	2.97	2.75	3.00	2.31	2.69	1.81	2.36
2.94	1.75	2.25	2.53	4.11	4.11	3.50	2.97	1.17	1.92	1.39	2.47
3.22	2.06	1.69	1.83	3.72	2.28	2.89	1.47	3.22	4.92	3.31	3.08
1.11	1.25	2.08	2.64	2.33	2.83	1.92	3.86	2.97	2.75	1.64	2.69
2.58	2.92	2.03	2.69	3.44	3.44	3.72	3.39	3.94	4.22	5.19	3.31
3.86	4.06	4.81	5.72	6.75	3.86	2.22	2.14	2.14	2.42	2.69	2.69
2.28	2.00	2.64	3.28	2.97	2.86	3.00	4.53	4.64	3.92	3.39	1.86
1.03	1.17	1.28	2.31	2.25	1.69	1.58	1.22	2.67	2.03	1.69	1.97
2.56	2.53	2.22	2.67	0.58	0.22	3.19	0.83	2.00	2.97	2.06	1.25
0.92	1.44	1.92	2.08	2.17	2.50	3.17	3.22	2.97	4.06	1.72	2.25
2.03	3.61	2.61	1.47	2.03	2.86	1.64	0.47	0.14	6.67	3.36	2.53
3.06	3.47	4.08	2.64	2.11	3.53	3.78	4.81	4.14	1.14	1.69	1.75
1.86	1.69	1.75	1.58	1.53	1.36	1.36	2.17	2.67	4.06	4.28	2.92
2.22											

4.2: Network Traffic Data

Data 1 (Rxx = 14175.3,σ = 55.81)

119.06	111.06	108.83	105.17	91.17	107.03	122.39	143.25	155.94	158.97	165.36
165.92	127.33	174.08	166.92	151.17	144.17	124.00	147.86	156.94	171.50	177.00
182.31	185.42	185.83	173.17	211.03	189.86	186.31	183.64	190.67	183.64	186.17
215.44	220.47	263.56	322.25	342.94	306.97	223.78	217.53	210.31	215.22	244.56
226.58	242.97	265.22	272.06	299.33	313.03	304.58	213.47	162.28	154.47	152.81
139.97	154.17	151.56	160.53	158.67	176.00	212.75	236.78	225.50	138.67	127.86
119.83	115.44	138.47	116.17	117.03	135.56	144.31	184.22	230.19	220.50	126.50
123.06	123.97	117.22	114.00	132.72	114.86	115.19	119.69	119.86	110.53	102.94
103.56	98.47	196.28	190.86	189.97	186.58	206.44	183.42	187.22	203.64	214.67
259.75	287.03	282.72	231.36	131.89	122.11	118.19	113.53	150.14	121.83	125.39
137.69	130.78	147.39	157.89	138.22	102.50	192.31	182.33	172.69	162.75	214.61
177.22	182.50	205.47	225.31	246.83	305.03	324.00	278.94	146.36	148.06	140.14
148.81	142.36	145.28	165.50	176.69	183.22	202.19	234.11	231.33	180.50	

Data 2 (Rxx = 125713,σ = 100.74)

354.56	320.47	307.17	289.64	304.86	281.31	329.69	364.00	397.25	478.00	569.53
501.14	346.78	275.94	263.56	267.03	254.11	206.92	244.11	241.14	279.58	306.58
393.42	498.61	535.08	448.58	295.31	288.28	283.81	262.89	250.44	269.17	304.11
335.03	331.50	397.28	444.97	396.92	258.86	252.61	234.17	231.69	235.97	243.86
239.81	262.92	280.56	290.72	316.78	325.53	308.11	238.11	301.31	292.06	289.39
267.72	252.67	258.03	274.69	313.67	355.75	432.06	548.22	561.22	428.42	380.03
371.00	364.06	351.78	338.50	357.00	381.61	432.03	460.92	533.39	649.50	691.47
613.97	387.75	371.39	361.31	354.53	334.56	367.94	374.78	434.25	445.78	521.81
647.53	689.67	603.53	223.78	217.53	210.31	215.22	244.56	226.58	242.97	265.22
272.06	299.33	313.03	304.58	213.47	392.25	403.92	396.86	369.92	357.72	354.36
394.03	379.36	412.36	474.00	495.14	446.89	345.22	328.06	327.11	320.83	333.86
316.75	332.14	364.97	366.47	425.75	498.81	470.06	362.67	280.31	256.11	237.83
223.31	219.97	236.94	276.53	287.00	340.39	427.39	420.72	345.64	329.36	310.33
298.72	285.03	274.00	283.86	328.00	372.86	398.72	297.92	264.78	252.83	246.00
263.33	236.28	239.97	278.72	301.83	346.69	413.00	399.75	315.67		

Data 3 (Rxx = 450832,σ = 97.08)

671.44	647.06	627.64	639.94	704.00	632.44	656.14	681.56	673.33	721.14	808.61
808.03	679.44	640.14	595.17	552.67	520.56	590.58	522.61	555.08	629.36	673.94
786.19	574.72	789.47	807.42	652.97	742.17	708.11	659.08	609.44	580.75	645.17
715.44	776.22	651.42	570.31	408.81	911.39	828.14	798.81	783.75	760.56	683.92
641.83	612.50	775.25	664.06	690.69	776.28	606.08	571.44	540.86	528.92	531.61
489.89	526.94	567.72	601.58	699.14	806.89	705.64	623.33	657.19	548.83	854.50
791.14	722.06	678.44	664.94	770.69	706.50	682.94	640.67	619.00	613.69	745.94
747.39	767.97	679.14	642.61	899.75	864.78	803.31	749.36	727.83	766.92	650.42
616.03	614.28	588.14	541.25	617.00	657.08	762.69	788.56			

APPENDIX 5: TABLES OF RESULTS

Acceptable rms error for all data sets was set at **20%**.

5.1: Exprerimental Data Results

Table 6.1: Table of filter efficiencies for data set 1 (Rxx = 0.0112551, σ = 0.28)

Table of filter efficiencies	P_0 = 2.5	P_0 = 70	Q_0 = 0.001	Q_0 = 0.1	R = 0.006	R = 0.2
n = 2	*4*	*5*	*6*	*4*	*6*	*4*
n = 5	*4*	*3*	*3*	*6*	*4*	*1*
n = 10	*8*	*7*	*3*	*8*	*11*	*3*

Table 6.2: Table of filter efficiencies for data set 2 (Rxx = 1, σ = 0.71)

Table of filter efficiencies	P_0 = 2.5	P_0 = 70	Q_0 = 0.001	Q_0 = 0.1	R = 0.006	R = 0.2
n = 2	*21*	*21*	*22*	*22*	*16*	*21*
n = 5	*36*	*36*	*33*	*37*	*36*	*36*
n = 10	*35*	*34*	*34*	*33*	*34*	*34*

Table 6.3: Table of filter efficiencies for data set 3 (Rxx = 4.1209, σ = 1.14)

Table of filter efficiencies	P_0 = 2.5	P_0 = 70	Q_0 = 0.001	Q_0 = 0.1	R = 0.006	R = 0.2
n = 2	*20*	*18*	*21*	*19*	*21*	*20*
n = 5	*39*	*38*	*37*	*39*	*39*	*37*
n = 10	*43*	*42*	*43*	*43*	*43*	*43*

5.2: Network Traffic Data Results

Table 6.4: Table of filter efficiencies for network traffic data

Network Traffic Data	Rxx	σ	n	P_0	Q_0	R	Filter Efficiency (%)
Data 1	*14175.3*	55.81	10	*2.5*	*0.151853*	*0.015*	*85*
Data 2	*125713*	100.74	15	*2.5*	*0.025*	*0.006*	*86*
Data 3	*450832*	97.08	13	*2.5*	*0.0185553*	*0.25*	*88*