A Project Report

TARGET TRACKING USING KALMAN FILTER

Submitted for partial fulfilment of the requirements for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

by

Sanjana Sambur (160115733143)

Under the guidance of

Mr. R. Srikanth

Asst. Professor

Department of Computer Science and Engineering,
Chaitanya Bharathi Institute of Technology (Autonomous),
(Affiliated to Osmania University, Hyderabad)
Hyderabad, TELANGANA (INDIA) – 500 075
May-2019

CERTIFICATE

This is to certify that the project titled "**Target Tracking using Kalman Filter**" is the bonafide work carried out by **Sanjana Sambur (160115733413)**, a student of B.E.(CSE) of Chaitanya Bharathi Institute of Technology, Hyderabad, affiliated to Osmania University, Hyderabad, Telangana(India) during the academic year 2018-19, submitted in partial fulfillment of the requirements for the award of the degree in **Bachelor of Engineering** (**Computer Science and Engineering**) and that the project has not formed the basis for the award previously of any other degree, diploma, fellowship or any other similar title.

Supervisor:	**Head, CSE Dept:**
Mr. R Srikanth,	**Dr. M Swamy Das**
Asst. Professor	

Place: Hyderabad
Date:

External Examiner

DECLARATION

We hereby declare that the research work entitled **TARGET TRACKING USING KALMAN FILTER** is original and bonafide work carried out by us as a part of fulfilment for Bachelor of Engineering in Computer Science and Engineering, Chaitanya Bharathi Institute of Technology, Gandipet, Hyderabad, under the guidance of Mr. R Srikanth, Asst. Professor, Department of CSE, CBIT.

No part of the project work is copied from books/journals/internet and wherever the partition is taken, the same has been duly referred in the text. The reported are based on the project work done entirely by us and not copied from any other source.

Sanjana Sambur
160115733143

Place: Hyderabad
Date:

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without introducing the people who made it possible and whose constant guidance and encouragement crowns all efforts with success. They have been a guiding light and source of inspiration towards the completion of the project.

We would like to express our sincere gratitude and indebtedness to our Mr. R Srikanth, who has supported us throughout our project with patience and knowledge.

We are also thankful to Head of the department, Dr. M Swamy Das for providing excellent infrastructure and a conducive atmosphere for completing this project successfully.

We are also extremely thankful to our Project Coordinator Dr. K. Sagar, Professor Dept. of CSE and Dr. M. Venu Gopalachari, Associate Professor, Dept. of CSE, for his valuable suggestions and interest throughout the course of this project

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed.

Finally, we would like to take this opportunity to thank our families for their support through the work. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Sanjana Sambur

160115733143

ABSTRACT

Object tracking is a crucial research subfield in computer vision and it has wide applications in navigation, robotics, military applications and space applications and so on. To estimate the undergoing movement, we use an approach called Extended Kalman filtering. Experimental results show that the Extended Kalman Filter (EKF) has the advantages of high efficiency, high robustness and high precision when tested and validated on different moving speed and different trajectories. Kalman filtering was very popular in the research field of navigation and aviation because of its magnificent accurate estimation characteristic. Since then, electrical engineers manipulate its advantages to useful purpose in target tracking systems. Consequently, today it had become a popular filtering technique for estimating and resolving redundant errors involves in tracing the target. However, unsatisfying tracking results may be produced due to different real-time conditions. During tracking, features are automatically selected from the input device (MPU6050). For each input, an estimated observation and multiple perturbed observations are rendered for the object. Corresponding positions are extracted from the sample location, and their estimated/perturbed measurements are acquired. These sample measurements and the real measurements of the features are then sent to an extended EKF. Finally, the EKF uses the sample measurements to compute high order approximations of the nonlinear measurement functions, and updates the state estimate of the object in an iterative form. Thus, through this process we can track the motion of the object with approximate range.

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In 1960, R.E. Kalman published his famous paper describing a recursive solution to the discrete-data linear filtering problem. Since that time, the Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation. The Kalman filter is a mathematical power tool that is playing an increasingly important role in computer graphics as we include sensing of the real world in our systems.

1.1 Objective:

The main goal of my project is to get the lat longs and position of the target on map. Here, we try to obtain such positions when there are very low signals of internet. In such cases it is very helpful to get the location of the object at the nearest radius.

1.1.1 Definition

The Kalman Filter is an estimator for what is called the "linear quadratic problem", which focuses on estimating the instantaneous "state" of a linear dynamic system perturbed by white noise and other external disturbances.

1.1.2 Probabilistic Origins of the Filter

This section is a short section describing the justification as mentioned in the previous section for this justification is rooted in the probability of a priori estimate conditioned on all prior z k measurements (Bayes' rule). For now, it is sufficed to point out that the Kalman filter maintains the first two moments of the state distribution,

1.1.3 Scope

If the system is nonlinear is the question that arises in terms of the usage, since usage of nonlinear systems are responsible for noise and many other disturbances an extension for Kalman filter is obtained called as Extended Kalman filter which suits for even for nonlinear dynamical systems. It is further being used in training multilayer perceptron.

1.2 Problem Definition

Object tracking is a crucial research subfield in computer vision and it has wide applications in navigation, robotics, military applications and space applications and so on. To estimate the undergoing movement, we use an approach called Extended Kalman filtering.

1.3 Existing Systems

1.3.1 Kalman Filters

Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

There are very few developments based on these concepts. Presently, most systems use Extended Kalman Filters or Integrated Extended Kalman Filters. These are out dated.

1.3.2 Problems with existing system

Existing technologies for navigation are not without shortcomings. For example, global positioning system (GPS) based navigation systems require good signal reception from satellites in orbit. With a GPS-based system, navigation in urban areas and indoor navigation is sometimes compromised because of poor signal reception. In addition, GPS-based systems are unable to provide close-quarter navigation for vehicular accident avoidance. Other navigation systems suffer from sensor errors (arising from slippage, for example) and an inability to detect humans or other obstacles.

1.4 Proposed System

A device comprising at least location source to produce further location data along a trajectory of the electronic device within an environment, wherein the location data contains a plurality of features observed within the environment at a plurality of poses of the electronic device along the trajectory; an inertial measurement unit to produce motion data indicative of motion of the electronic device; and a processor configured to apply an Extended Kalman filter (EKF) as the electronic device traverses the trajectory, wherein the Extended Kalman filter is configured to:

1. Maintain a filter state vector storing estimates for a position at each of the poses along the trajectory and estimates for positions for one or more features within the environment
2. Compute, from the location data, one or more constraints based on features observed from multiple poses along the trajectory, and
3. Update, in accordance with the motion data and the one or more computed constraints, the estimates within the filter state vector of the Extended Kalman filter while excluding, from the filter state vector, state estimates for positions within the environment for the features that were observed from the multiple poses and for which the one or more constraints were computed.

The device wherein extended Kalman filter is configured to store in the filter state vector estimates for an orientation at each of the poses along the trajectory and to update the estimates in accordance with the motion data and the one or more computed constraints.
The device wherein the processor is configured to generate each of the one or more constraints by computing a residual of a measurement for the respective feature.
With additional integration with Wi-Fi module, made with arduino one will be able to send and receive messages where there are no signals from satellites. This can be further

integrated with the Google maps to obtain the exact path of the object. Later this information can be utilized for guiding the path.

1.5.1 Applications

1.5.1.1 GPS-Denied Navigation

GPS-denied navigation for small unmanned aircraft systems (UAS) is an active and rich field of research with significant practical applications such as infrastructure inspection and security.

1.5.1.2 Satellite Navigation or Satnav System

A satellite navigation or satnav system is a system that uses satellites to provide autonomous geo-spatial positioning. It allows small electronic receivers to determine their location (longitude, latitude, and altitude/elevation) to high precision (within a few metres) using time signals transmitted along a line of sight by radio from satellites. The system can be used for providing position, navigation or for tracking the position of something fitted with a receiver (satellite tracking).

1.5.1.3 Attitude and Heading Reference System (AHRS)

An attitude and heading reference system (AHRS) consists of sensors on three axes that provide attitude information for aircraft, including roll, pitch and yaw. These are sometimes referred to as MARG (Magnetic, Angular Rate, and Gravity) sensors and consist of either solid-state or micro electro mechanical systems (MEMS) gyroscopes, accelerometers and magnetometers. They are designed to replace traditional mechanical gyroscopic flight instruments.

1.5.1.4 Radar Tracker

A radar tracker is a component of a radar system, or an associated command and control (C2) system, that associates consecutive radar observations of the same target into tracks. It is particularly useful when the radar system is reporting data from several different targets or when it is necessary to combine the data from several different radars or other sensors.

1.5.1.5 Navigation System

A navigation system is a (usually electronic) system that aids in navigation. Navigation systems may be entirely on board a vehicle or vessel (on the ships) bridge, or they may be located elsewhere and communicate via radio or other signals with a vehicle or vessel, or they may use a combination of these methods.

- containing maps, which may be displayed in human readable format via text or in a graphical format
- determining a vehicle or vessel's location via sensors, maps, or information from external sources
- providing suggested directions to a human in charge of a vehicle or vessel via text or speech
- providing directions directly to an autonomous vehicle such as a robotic probe or guided missile
- providing information on nearby vehicles or vessels, or other hazards or obstacles
- providing information on traffic conditions and suggesting alternative directions

CHAPTER 2

LITERATURE SURVEY

Introduction

Localization and navigation systems and techniques are described. An electronic device comprises a processor configured to apply an extended Kalman filter (EKF) as the electronic device traverses the trajectory. The extended Kalman filter is configured to maintain a state vector storing estimates for a position of the electronic device at poses along a trajectory within an environment along with estimates for positions for one or more features within the environment. The EKF computes constraints based on features observed from multiple poses along the trajectory, and updates, in accordance with motion data and the one or more computed constraints, the estimates within the state vector of the extended Kalman filter while excluding, from the state vector, state estimates for positions within the environment for the features that were observed from the multiple poses and for which the constraints were computed.

2.1 EKF (Extended Kalman Filter)

In estimation theory, the extended Kalman filter (EKF) is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance. In the case of well-defined transition models, the EKF has been considered the de facto standard in the theory of nonlinear state estimation, navigation systems and GPS.

2.1.1 Formulation

In the extended Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$\boldsymbol{x}_k = f(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k$$

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

2.1.2 Continuous-time extended Kalman Filter

Model

$$\dot{\mathbf{x}}(t) = f\big(\mathbf{x}(t), \mathbf{u}(t)\big) + \mathbf{w}(t) \qquad \mathbf{w}(t) \sim \mathcal{N}\big(\mathbf{0}, \mathbf{Q}(t)\big)$$
$$\mathbf{z}(t) = h\big(\mathbf{x}(t)\big) + \mathbf{v}(t) \qquad \mathbf{v}(t) \sim \mathcal{N}\big(\mathbf{0}, \mathbf{R}(t)\big)$$

Initialize

$$\hat{\mathbf{x}}(t_0) = E\big[\mathbf{x}(t_0)\big], \mathbf{P}(t_0) = Var\big[\mathbf{x}(t_0)\big]$$

Predict-Update

$$\dot{\hat{\mathbf{x}}}(t) = f\big(\hat{\mathbf{x}}(t), \mathbf{u}(t)\big) + \mathbf{K}(t)\Big(\mathbf{z}(t) - h\big(\hat{\mathbf{x}}(t)\big)\Big)$$
$$\dot{\mathbf{P}}(t) = \mathbf{F}(t)\mathbf{P}(t) + \mathbf{P}(t)\mathbf{F}(t)^\top - \mathbf{K}(t)\mathbf{H}(t)\mathbf{P}(t) + \mathbf{Q}(t)$$
$$\mathbf{K}(t) = \mathbf{P}(t)\mathbf{H}(t)^\top \mathbf{R}(t)^{-1}$$
$$\mathbf{F}(t) = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}(t), \mathbf{u}(t)}$$
$$\mathbf{H}(t) = \left.\frac{\partial h}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}(t)}$$

Unlike the discrete-time extended Kalman filter, the prediction and update steps are coupled in the continuous-time extended Kalman filter.

The above recursion is a first-order extended Kalman filter (EKF). Higher order EKFs may be obtained by retaining more terms of the Taylor series expansions. For example, second and third order EKFs have been described. However, higher order EKFs tend to only provide performance benefits when the measurement noise is small.

It this important to state that the EKF is not an optimal filter, but rather it is implemented based on a set of approximations. Thus, the matrices P(k|k) and P(k + 1|k) do not represent the true covariance of the state estimates. Moreover, as the matrices F(k) and H(k) depend on previous state estimates and therefore on measurements, the filter gain K(k) and the matrices P(k|k) and P(k + 1|k) cannot be computed off-line as occurs in the Kalman filter. Contrary to the Kalman filter, the EKF may diverge, if the consecutive linearizations are not a good approximation of the linear model in all the associated uncertainty domain.

Mathematical calculations:

Prediction

Assume that $p(x_k \mid Y_1^k)$ is a Gaussian pdf with mean η_F^n [1] and covariance matrix V_F^n, i.e.,

$$p(x_k \mid Y_1^k) \sim \mathcal{N}(x_k - \eta_F^k, V_F^k) = \mathcal{N}(x_k - \hat{x}(k|k), P(k|k)).$$

From the non-linear system dynamics,

$$x_{k+1} = f_k(x_k) + w_k,$$

and the Bayes law, the conditional pdf of x_{k+1} given Y_1^k is given by

$$p(x_{k+1} \mid Y_1^k) = \int_{-\infty}^{\infty} p(x_{k+1} \mid x_k) p(x_k \mid Y_1^k) dx_k,$$

or also,

$$p(x_{k+1} \mid Y_1^k) = \int_{-\infty}^{\infty} p_{w_k}(x_{k+1} - f_k(x_k)) p(x_k \mid Y_1^k) dx_k$$

where

$$p_{w_k}(x_{k+1} - f_n(x_k)) = \frac{1}{(2\pi)^{n/2}[detQ_k]^{1/2}} \exp[-\frac{1}{2}(x_{k+1} - f_k(x_k))^T Q_k^{-1}(x_{k+1} - f_k(x_k))].$$

The previous expression is **not** a Gaussian pdf given the nonlinearity in x_k. We will linearize $f_k(x_k)$ in (5.6) around $\eta_F^k = \hat{x}(k \mid k)$ negleting higher order terms, this yielding

$$\begin{aligned} f_k(x_k) &\cong f_k(\eta_F^k) + \nabla f_k \mid_{\eta_F^k} \cdot [x_k - \eta_F^k] \\ &= \overbrace{f_k(\eta_F^k) - \nabla f_k \mid_{\eta_F^k} \cdot \eta_F^k}^{s_k} + \nabla f_k \mid_{\eta_F^k} \cdot x_k. \end{aligned}$$

where ∇f_k is the Jacobian matrix of $f(.)$,

$$\nabla f_k = \frac{\partial f(x(k))}{\partial x(k)} \mid_{\eta_F^k}$$

With this linearization, the system dynamics may be written as:

$$x_{k+1} = \bigtriangledown f_k \mid_{\eta_F^k} \cdot x_k + w_k + \underbrace{[f_k(\eta_F^k) - \bigtriangledown f_k \mid_{\eta_F^k} \cdot \eta_F^k]}_{s_k}$$

or, in a condensed format,

$$\boxed{x_{k+1} = \bigtriangledown f_k \mid_{\eta_F^k} \cdot x_k + w_k + s_k}$$

Note that (5.11) represents a linear dynamics, in which s_k is known, has a null conditional expected value and depends on previous values of the state estimate. According to (5.9) the pdf in (5.7) can be written as:

$$\begin{aligned} p(x_{k+1} \mid Y_1^k) &= \int_{-\infty}^{\infty} p_{w_k}(x_{k+1} - \bigtriangledown f_k \mid_{\eta_F^k} \cdot x_k - s_k) \cdot p(x_k \mid Y_1^k) dx_k \\ &= \int_{-\infty}^{\infty} \mathcal{N}(x_{k+1} - \bigtriangledown f_k \mid_{\eta_F^k} \cdot x_k - s_k, Q_k) \cdot \mathcal{N}(x_k - \eta_F^k, V_F^k) dx_k \\ &= \int_{-\infty}^{\infty} \mathcal{N}(x_{k+1} - s_k - \bigtriangledown f_k \mid_{\eta_F^k} \cdot x_k, Q_k) \mathcal{N}(x_k - \eta_F^k, V_F^k) dx_k \end{aligned}$$

To simplify the computation of the previous pdf, consider the following variable transformation

$$z_k = \bigtriangledown f_k \cdot x_k.$$

where we considered, for the sake of simplicity, the simplified notation $\bigtriangledown f_k$ to represent $\bigtriangledown f_k \mid_{\eta_F^k}$.

Evaluating the mean and the covariance matrix of the random vector results:

$$\begin{aligned} E[y_k] &= \bigtriangledown f_k \cdot E[x_k] = \bigtriangledown f_k \cdot \eta_F^k \\ E[y_k y_k^T] &= \bigtriangledown f_k \cdot V_F^k \cdot \bigtriangledown f_k^T. \end{aligned}$$

From the previous result, the pdf of x_k in (5.5) may be written as:

$$\begin{aligned} &\mathcal{N}(x_k - \eta_F^k, V_F^k) = \\ &\quad \frac{1}{(2\pi)^{n/2}(detV_F^k)^{1/2}} exp[-\frac{1}{2}(x_k - \eta_F^k)^T (V_F^k)^{-1}(x_n - \eta_F^k)] = \\ &\quad \frac{1}{(2\pi)^{n/2}(detV_F^k)^{1/2}} exp[-\frac{1}{2}(\bigtriangledown f_k x_k - \bigtriangledown f_k \cdot \eta_F^k)^T (\bigtriangledown f_F^k)^{-T} (V_F^k)^{-1} (\bigtriangledown f_F^k)^{-1} (\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k)] = \\ &\quad \frac{1}{(2\pi)^{n/2}(detV_F^k)^{1/2}} exp[-\frac{1}{2}(\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k)^T (\bigtriangledown f_k \cdot V_F^k \bigtriangledown f_k^T)^{-1} (\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k)] = \\ &\quad = det \bigtriangledown f_k \cdot \frac{1}{(2\pi)^{n/2}(det \bigtriangledown f_k V_F^k \bigtriangledown f_k^T)^{n/2}} \\ &\quad exp[-\frac{1}{2}(\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k)^T (\bigtriangledown f_k V_F^k \bigtriangledown f_k^T)^{-1} (\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k)]. \end{aligned}$$

We thus conclude that

$$\mathcal{N}(x_k - \eta_F^k, V_F^k) = det \bigtriangledown f_k \cdot \mathcal{N}(\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k, \bigtriangledown f_k V_F^k \bigtriangledown f_k^T).$$

Replacing (5.16) in (5.12) yields:

$$\begin{aligned} &p(x_{k+1} \mid Y_1^k) = \\ &= \int_{-\infty}^{\infty} \mathcal{N}(x_{k+1} - s_k - \bigtriangledown f_k x_k, Q_k) \mathcal{N}(\bigtriangledown f_k x_k - \bigtriangledown f_k \eta_F^k, \bigtriangledown f_k V_F^k \bigtriangledown f_k^T) d(\bigtriangledown f_k \cdot x_k) \\ &\qquad = \mathcal{N}(x_{k+1} - s_k, Q_k) \star \mathcal{N}(x_{k+1} - \bigtriangledown f_k \cdot \eta_F^k, \bigtriangledown f_k V_F^k \bigtriangledown f_k^T) \end{aligned}$$

where $\star$ represents the convolution of the two functions. We finally conclude that,

$$p(x_{k+1} \mid Y_1^k) = \mathcal{N}(x_{k+1} - \bigtriangledown f_k \mid_{\eta_F^k} \cdot \eta_F^k - s_k, Q_k + \bigtriangledown f_k \mid_{\eta_F^k} V_F^k \bigtriangledown f_k \mid_{\eta_F^k}^T$$

We just conclude that,

if $p(x_k \mid Z_1^k)$ is a Gaussian pdf with

1. mean η_F^n,
2. covariance matrix V_F^n

then, the linearization of the dynamics around η_F^n yields $p(x_{k+1} \mid Z_1^k)$, which is a Gaussian pdf with

1. mean η_P^{k+1}
2. covariance matrix V_P^{k+1}

where

$$\eta_P^{k+1} = \bigtriangledown f_k \mid_{\eta_F^k} \cdot \eta_F^k + f_k(\eta_F^k) - \bigtriangledown f_k \mid_{\eta_F^k} \cdot \eta_F^k$$

or else, given the value of s_k given in (5.10), can be simplified to

$$\begin{aligned} \eta_P^{k+1} &= f_k(\eta_F^k) \\ V_P^{k+1} &= Q_k + \bigtriangledown f_k \mid_{\eta_F^k} \cdot V_F^k \cdot \bigtriangledown f_k^T \mid_{\eta_F^k} . \end{aligned}$$

These values are taken as the predicted state estimate and the associated covariance obtained by the EKF, i.e.,

$$\begin{aligned} \hat{x}(k+1|k) &= \eta_P^{k+1} \\ P(k+1|k) = &= V_P^{k+1}, \end{aligned}$$

$$\begin{aligned} \hat{x}(k+1|k) &= f_k(\hat{x}(k|k)) \\ P(k+1|k) &= \bigtriangledown f_k \mid_{\eta_F^k} \cdot P(k|k) \cdot \bigtriangledown f_k^T \mid_{\eta_F^k} \end{aligned}$$

2.2 Terminologies

2.2.1 Latitude

The "latitude" (abbreviation: Lat., φ, or phi) of a point on Earth's surface is the angle between the equatorial plane and the straight line that passes through that point and through (or close to) the center of the Earth. Lines joining points of the same latitude trace circles on the surface of Earth called parallels, as they are parallel to the Equator and to each other. The North Pole is 90° N; the South Pole is 90° S. The 0° parallel of latitude is designated the Equator, the fundamental plane of all geographic coordinate systems. The Equator divides the globe into Northern and Southern Hemispheres.

2.2.2 Longitude

The "longitude" (abbreviation: Long., λ, or lambda) of a point on Earth's surface is the angle east or west of a reference meridian to another meridian that passes through that point. All meridians are halves of great ellipses (often called great circles), which converge at the North and South Poles. The prime meridian determines the proper Eastern and Western Hemispheres, although maps often divide these hemispheres further west in order to keep the Old World on a single side. The antipodal meridian of Greenwich is both 180°W and 180°E.

The combination of these two components specifies the position of any location on the surface of Earth, without consideration of altitude or depth. The grid formed by lines of latitude and longitude is known as a "graticule".

2.2.3 Map Projection

To establish the position of a geographic location on a map, a map projection is used to convert geodetic coordinates to plane coordinates on a map; it projects the datum ellipsoidal coordinates and height onto a flat surface of a map. The datum, along with a map projection applied to a grid of reference locations, establishes a grid system for plotting locations. Common map projections in current use include the Universal Transverse Mercator (UTM), the Military Grid Reference System (MGRS), the United

States National Grid (USNG), the Global Area Reference System (GARS) and the World Geographic Reference System (GEOREF).[10] Coordinates on a map are usually in terms northing N and easting E offsets relative to a specified origin.

Map projection formulas depend in the geometry of the projection as well as parameters dependent on the particular location at which the map is projected. The set of parameters can vary based on type of project and the conventions chosen for the projection. For the transverse Mercator projection used in UTM, the parameters associated are the latitude and longitude of the natural origin, the false northing and false easting, and an overall scale factor.[11] Given the parameters associated with particular location or grin, the projection formulas for the transverse Mercator are a complex mix of algebraic and trigonometric functions.

2.3 Advantages

2.3.1

The Extended Kalman filter algorithm is implementable on a digital computer, which this was replaced by analog circuitry for estimation and control when Kalman filter was first introduced. This implementation may be slower compared to analog filters of Wiener; however, it is capable of much greater accuracy.

2.3.2

Stationary properties of the Extended Kalman filter are not required for the deterministic dynamics or random processes. Many applications of importance include non-stationary stochastic processes.

2.3.3

The Extended Kalman filter is compatible with state space formulation of optimal controllers for dynamic systems. It proves useful towards the two properties of estimation and control for these systems.

a) It requires less additional mathematical preparation to learn for the modern control engineering student, compared to the Wiener filter.

b) Necessary information for mathematically sound, statistically-based decision methods for detecting and rejecting anomalous measurements are provided through the use of Extended Kalman filter.

2.3.4

It can be used for non-linear applications which can eradicate maximum amount of external disturbances.

CHAPTER 3
METHODOLOGY

INTRODUCTION

In statistics and control theory, Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe. The filter is named after Rudolf E. Kálmán, one of the primary developers of its theory.

The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft.[1] Furthermore, the Kalman filter is a widely applied concept in time series analysis used in fields such as signal processing and econometrics. Kalman filters also are one of the main topics in the field of robotic motion planning and control, and they are sometimes included in trajectory optimization. The Kalman filter also works for modeling the central nervous system's control of movement. Due to the time delay between issuing motor commands and receiving sensory feedback, use of the Kalman filter supports a realistic model for making estimates of the current state of the motor system and issuing updated commands.[2]

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

3.1 SYSTEM DESIGN

3.1.1 Proposed Algorithm

The Kalman filter uses a system's dynamic model (e.g., physical laws of motion), known control inputs to that system, and multiple sequential measurements (such as from sensors) to form an estimate of the system's varying quantities (its state) that is better than the estimate obtained by using only one measurement alone. As such, it is a common sensor fusion and data fusion algorithm.

Noisy sensor data, approximations in the equations that describe the system evolution, and external factors that are not accounted for all place limits on how well it is possible to determine the system's state. The Kalman filter deals effectively with the uncertainty due to noisy sensor data and to some extent also with random external factors. The Kalman filter produces an estimate of the state of the system as an average of the system's predicted state and of the new measurement using a weighted average. The purpose of the weights is that values with better (i.e., smaller) estimated uncertainty are "trusted" more. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone. This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

The relative certainty of the measurements and current state estimate is an important consideration, and it is common to discuss the response of the filter in terms of the Kalman filter's *gain*. The Kalman gain is the relative weight given to the measurements and current state estimate, and can be "tuned" to achieve particular performance. With a high gain, the filter places more weight on the most recent measurements, and thus follows them more responsively. With a low gain, the filter follows the model predictions more closely. At the extremes, a high gain close to one will result in a jumpier estimated trajectory, while low gain close to zero will smooth out noise but decrease the responsiveness.

When performing the actual calculations for the filter (as discussed below), the state estimate and covariances are coded into matrices to handle the multiple dimensions involved in a single set of calculations. This allows for a representation of linear relationships between different state variables (such as position, velocity, and acceleration) in any of the transition models or covariances.

The Kalman filter is a recursive estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. In contrast to batch estimation techniques, no history of observations and/or estimates is required. In what follows, the notation $x_{n|m}$ at time *n* given observations up to and including at time $m \leq n$.

The state of the filter is represented by two variables:

- $x_{k|k}$, the *a posteriori* state estimate at time *k* given observations up to and including at time *k*;
- $P_{k|k}$, the *a posteriori* error covariance matrix (a measure of the estimated accuracy of the state estimate).

The Kalman filter can be written as a single equation, however it is most often conceptualized as two distinct phases: "Predict" and "Update". The predict phase uses the state estimate from the previous timestep to produce an estimate of the state at the current timestep. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current timestep, it does not include observation information from the current timestep. In the update phase, the current *a priori* prediction is combined with current observation information to refine the state estimate. This improved estimate is termed the *a posteriori* state estimate.

Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation. However, this is not necessary; if an observation is unavailable for some reason, the update may be skipped and multiple prediction steps performed. Likewise, if multiple independent observations are available at the same time, multiple update steps may be performed (typically with different observation matrices $\mathbf{H}_k$).

3.1.1.1 Predict

Predict (a priori) state estimate $\hat{\mathbf{x}}_{\mathbf{k|k-1}} = \mathbf{F_k x_{k-1|k-1}} + \mathbf{B_k u_k}$

Predict (a priori) error covariance $\mathbf{P_{k|k-1}} = \mathbf{F_k P_{k|k-1} F_k^T} + \mathbf{Q_k}$

3.1.1.2 Update

Innovation or measurement pre-fit residual $\tilde{\mathbf{y}}_{\mathbf{k}} = \mathbf{z_k} - \mathbf{H_k}\hat{\mathbf{x}}_{\mathbf{k|k-1}}$

Innovation (or pre-fit residual) covariance $\mathbf{S_k} = \mathbf{R_k} + \mathbf{H_k P_{k|k-1} H_k^T}$

Optimal Kalman gain $\mathbf{K_k} = \mathbf{P_{k|k-1} H_k^{-1} S_k^{-1}}$

Updated (*a posteriori*) state estimate $\hat{\mathbf{x}}_{\mathbf{k|k-1}} = \mathbf{x_{k-1|k-1}} + \mathbf{K_k}\tilde{\mathbf{y}}_{\mathbf{k}}$

Updated (*a posteriori*) estimate covariance $\mathbf{P_{k|k}} = (\mathbf{I} - \mathbf{K_k H_k})\mathbf{P_{k|k-1}}(\mathbf{I} - \mathbf{K_k H_k})^{\mathbf{T}} + \mathbf{K}$

Measurement post-fit residual $\tilde{\mathbf{y}}_{\mathbf{k|k}} = \mathbf{z_k} - \mathbf{H_k}\hat{\mathbf{x}}_{\mathbf{k|k}}$

The formula for the updated (*a posteriori*) estimate covariance above is valid for any gain $\mathbf{K}_k$ and is sometimes called the **Joseph form**. For the optimal Kalman gain the formula further simplifies to $\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\,\mathbf{P}_{k|k-1}$, in which form it is most widely used in applications. However, one must keep in mind, that it is valid only for the optimal gain that minimizes the residual error. Proof of the formulae is found in the *derivations* section.

3.1.1.3 Invariants

If the model is accurate, and the values for $\mathbf{x_{0|0}}$ and $\mathbf{P_{0|0}}$ accurately reflect the distribution of the initial state values, then the following invariants are preserved:

$$\mathbf{E}\left[\mathbf{x_k} - \hat{\mathbf{x}}_{\mathbf{k|k}}\right] = \mathbf{E}\left[\mathbf{x_k} - \hat{\mathbf{x}}_{\mathbf{k|k-1}}\right] = \mathbf{0}$$

$$\mathbf{E}\left[\mathbf{y}^{\sim}{}_{\mathbf{k}}\right] = \mathbf{0}$$

where E| ξ| is the expected value of ξ . That is, all estimates have a mean error of zero.

Also:

$$\mathbf{p}_{\mathbf{k|k}} = \mathbf{cov}(\mathbf{x}_{\mathbf{k}} - \hat{\mathbf{x}}_{\mathbf{k|k}})$$

$$\mathbf{p}_{\mathbf{k|k-1}} = \mathbf{cov}(\mathbf{x}_{\mathbf{k}} - \hat{\mathbf{x}}_{\mathbf{k|k-1}})$$

$$\mathbf{S}_{\mathbf{k}} = \mathbf{cov}([\tilde{\mathbf{y}}_{\mathbf{k}}])$$

So, covariance matrices accurately reflect the covariance of estimates.

3.1.1.4 Estimation of the noise covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$

Practical implementation of the Kalman Filter is often difficult due to the difficulty of getting a good estimate of the noise covariance matrices $\mathbf{Q}_k$ and $\mathbf{R}_k$. Extensive research has been done in this field to estimate these covariances from data. One practical approach to do this is the *autocovariance least-squares (ALS)* technique that uses the time-lagged autocovariances of routine operating data to estimate the covariances. The GNU Octave and Matlab code used to calculate the noise covariance matrices using the ALS technique is available online under the GNU General Public License

3.1.1.5 Optimality and performance

It follows from theory that the Kalman filter is the optimal linear filter in cases where a) the model perfectly matches the real system, b) the entering noise is white (uncorrelated) and c) the covariances of the noise are exactly known. Several methods for the noise covariance estimation have been proposed during past decades, including ALS, mentioned in the section above. After the covariances are estimated, it is useful to evaluate the performance of the filter; i.e., whether it is possible to improve the state estimation quality. If the Kalman filter works optimally, the innovation sequence (the output prediction error) is a white noise, therefore the whiteness property of the innovation's measures filter performance. Several different methods can be used for this purpose. If the noise terms are non-Gaussian distributed, methods for assessing performance of the filter estimate, which use probability inequalities or large-sample theory, are known in the literature.

3.1.1.6 Deriving the *posteriori* estimate covariance matrix

Starting with our invariant on the error covariance $\mathbf{P}_{k\mid k}$ as above

$$\mathbf{p_{k|k} = cov(x_k - \hat{x}_{k|k})}$$

substitute in the definition of $\mathbf{x_{k|k}}$

and substitute $\mathbf{y_k}$

and $\mathbf{z_k}$

and by collecting the error vectors we get

$$\mathbf{p_{k|k} = cov((I - K_k H_k)(x_k - \hat{x}_{k|k-1}) - H_k v_k)}$$

Since the measurement error $\mathbf{v}_k$ is uncorrelated with the other terms, this becomes

$$\mathbf{p_{k|k} = cov\left((I - K_k H_k)(x_k - \hat{x}_{k|k-1})\right) + cov(H_k v_k)}$$

by the properties of vector covariance this becomes

|

$$\mathbf{p_{k|k}(I - K_k H_k) cov\left((x_k - \hat{x}_{k|k-1})\right)(I - K_k H_k)^T + K_k cov(v_k) {K_k}^T}$$

which, using our invariant on $\mathbf{P}_{k\mid k-1}$ and the definition of $\mathbf{R}_k$ becomes

$$\mathbf{p_{k|k} = (I - K_k H_k) p_{k|k-1} (I - K_k H_k)^T + K_k R_k {K_k}^T}$$

This formula (sometimes known as the **Joseph form** of the covariance update equation) is valid for any value of $\mathbf{K}_k$. It turns out that if $\mathbf{K}_k$ is the optimal Kalman gain, this can be simplified further as shown below.

3.1.1.7 Kalman gain derivation

The Kalman filter is a minimum mean-square error estimator. The error in the *a posteriori* state estimation is: $\boldsymbol{x_k - x_{k|k}}$

–

(

We seek to minimize the expected value of the square of the magnitude of this vector, $E[||\boldsymbol{x_k} - \boldsymbol{x_{k|k}}||^2]$This is equivalent to minimizing the trace of the *a posteriori* estimate covariance matrix .By expanding out the terms in the equation above and collecting, we get:

$$\mathbf{P_{k|k}} = \mathbf{P_{k|k-1}} - \mathbf{K_k H_k P_{k|k-1}} - \mathbf{P_{k|k-1}} - \mathbf{H_k^T K_k^T} + \mathbf{K_k(H_k P_{k|k-1} H_k^T + R_k)K_k^T}$$

$$= \mathbf{P_{k|k-1}} - \mathbf{K_k H_k P_{k|k-1}} - \mathbf{P_{k|k-1}} + \mathbf{K_k S_k K_k^T}$$

The trace is minimized when its matrix derivative with respect to the gain matrix is zero. Using the gradient matrix rules and the symmetry of the matrices involved we find that

$$\frac{\partial\ \mathrm{tr}(\mathbf{P}_{k|k})}{\partial\ \mathbf{K}_k} = -2\left(\mathbf{H}_k \mathbf{P}_{k|k-1}\right)^{\mathrm{T}} + 2\mathbf{K}_k \mathbf{S}_k = 0.$$

Solving this for $\mathbf{K}_k$ yields the Kalman gain:

$$\mathbf{K_k S_k} = (\mathbf{H_k P_{k|k-1}})^{\mathbf{T}} = \mathbf{P_{k|k-1} P_{k|k-1} H_k^T}$$

$$\mathbf{K_k} = \mathbf{P_{k|k-1} H_k^T P_{k|k-1} S_k^{-1}}$$

This gain, which is known as the *optimal Kalman gain*, is the one that yields MMSE estimates when used.

Simplification of the *posteriori* error covariance formula

The formula used to calculate the *a posteriori* error covariance can be simplified when the Kalman gain equals the optimal value derived above. Multiplying both sides of our Kalman gain formula on the right by $\mathbf{S}_k \mathbf{K}_k^{\mathrm{T}}$, it follows that

$$\mathbf{K_k S_k K_k^T} = \mathbf{P_{k|k-1} H_k^T K_k^T}$$

Referring back to our expanded formula for the *a posteriori* error covariance, we find the last two terms cancel out, giving

$$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1}}$$

This formula is computationally cheaper and thus nearly always used in practice, but is only correct for the optimal gain. If arithmetic precision is unusually low causing problems with numerical stability, or if a non-optimal Kalman gain is deliberately used, this simplification cannot be applied; the *a posteriori* error covariance formula as derived above (Joseph form) must be used.

Sensitivity Analysis

The Kalman filtering equations provide an estimate of the state $\mathbf{x_{k|k}}$ and its error covariance $\mathbf{P_{k|k}}$ recursively. The estimate and its quality depend on the system parameters and the noise statistics fed as inputs to the estimator. This section analyzes the effect of uncertainties in the statistical inputs to the filter.[27] In the absence of reliable statistics or the true values of noise covariance matrices $\mathbf{Q_k}$ and $\mathbf{R_k}$, the expression

$$\mathbf{P_{k|k} = (I - K_k H_k) P_{k|k-1} (I - K_k H_k)^T + K_k R_k K_k^T}$$

no longer provides the actual error covariance. In other words

$$\mathbf{P_{k|k} \neq E[\left(x_k - x_{(k|k)}\right)[(x_k - x_{k|k})^T]}$$

In most real-time applications, the covariance matrices that are used in designing the Kalman filter are different from the actual (true) noise covariances matrices. This sensitivity analysis describes the behavior of the estimation error covariance when the noise covariances as well as the system matrices $\mathbf{F_k}$ and $\mathbf{H_k}$ that are fed as inputs to the filter are incorrect. Thus, the sensitivity analysis describes the robustness (or sensitivity) of the estimator to mis specified statistical and parametric inputs to the estimator.

This discussion is limited to the error sensitivity analysis for the case of statistical uncertainties. Here the actual noise covariances are denoted by $\mathbf{Q_k^a}$and $\mathbf{R_k^a}$ respectively, whereas the design values used in the estimator are $\mathbf{Q_k}$ and $\mathbf{R_k}$ respectively. The actual error covariance is denoted by $\mathbf{P^a_{k|k}}$ and $\mathbf{P_{k|k}}$ as computed by the Kalman filter is referred to as the Riccati variable. When $\mathbf{Q_k} = \mathbf{Q_k^a}$ and, $\mathbf{R_k} = \mathbf{R_k^a}$ this means that $\mathbf{P_{k|k} = P^a_{k|k}}$. While computing the actual error covariance using $\mathbf{P}^a_{k|k} = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\right)\left(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k}\right)^\top\right]$, substituting for $\mathbf{x}_{k|k}$ and using the fact that $E\left[\mathbf{w}_k\mathbf{w}_k^\top\right] = \mathbf{Q}^a_k$ and $E\left[\mathbf{v}_k\mathbf{v}_k^\top\right] = \mathbf{R}^a_k$, results in the following recursive equations for

$\mathbf{P}^a_{k|k} : \mathbf{P}^a_{k|k-1} = \mathbf{F}_k\mathbf{P}^a_{k-1|k-1}\mathbf{F}_k^\top + \mathbf{Q}^a_k$ and $\mathbf{P}^a_{k|k} = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\mathbf{P}^a_{k|k-1}\left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)^\top + \mathbf{K}_k\mathbf{R}^a_k\mathbf{K}_k^\top$.

While computing $\mathbf{P}_{k|k}$, by design the filter implicitly assumes that $E\left[\mathbf{w}_k\mathbf{w}_k^\top\right] = \mathbf{Q}_k$ and $E\left[\mathbf{v}_k\mathbf{v}_k^\top\right] = \mathbf{R}_k$. Note that the recursive expressions for $\mathbf{P}^a_{k|k}$ and $\mathbf{P}_{k|k}$ are identical except for the presence of $\mathbf{Q}_k^a$ and $\mathbf{R}_k^a$ in place of the design values $\mathbf{Q}_k$ and $\mathbf{R}_k$ respectively. Researches have been done to analyze Kalman filter system's robustness.

3.1.2 Diagrammatic Representation

3.1.2.1 Activity Diagram:

The following picture shows the activity diagram of my project. Firstly, the sensor will take
Data from Gyroscope. We get raw data. Later we convert that to required format after few mathematical calculations. Send these values to extended Kalman Filter. This filter will improve the values by removing the noise and device errors. These values are then sent to the main program and this will now point the values on map.

Activity diagram below shows the flow of the process from the acquiring of data to the plotting of maps.

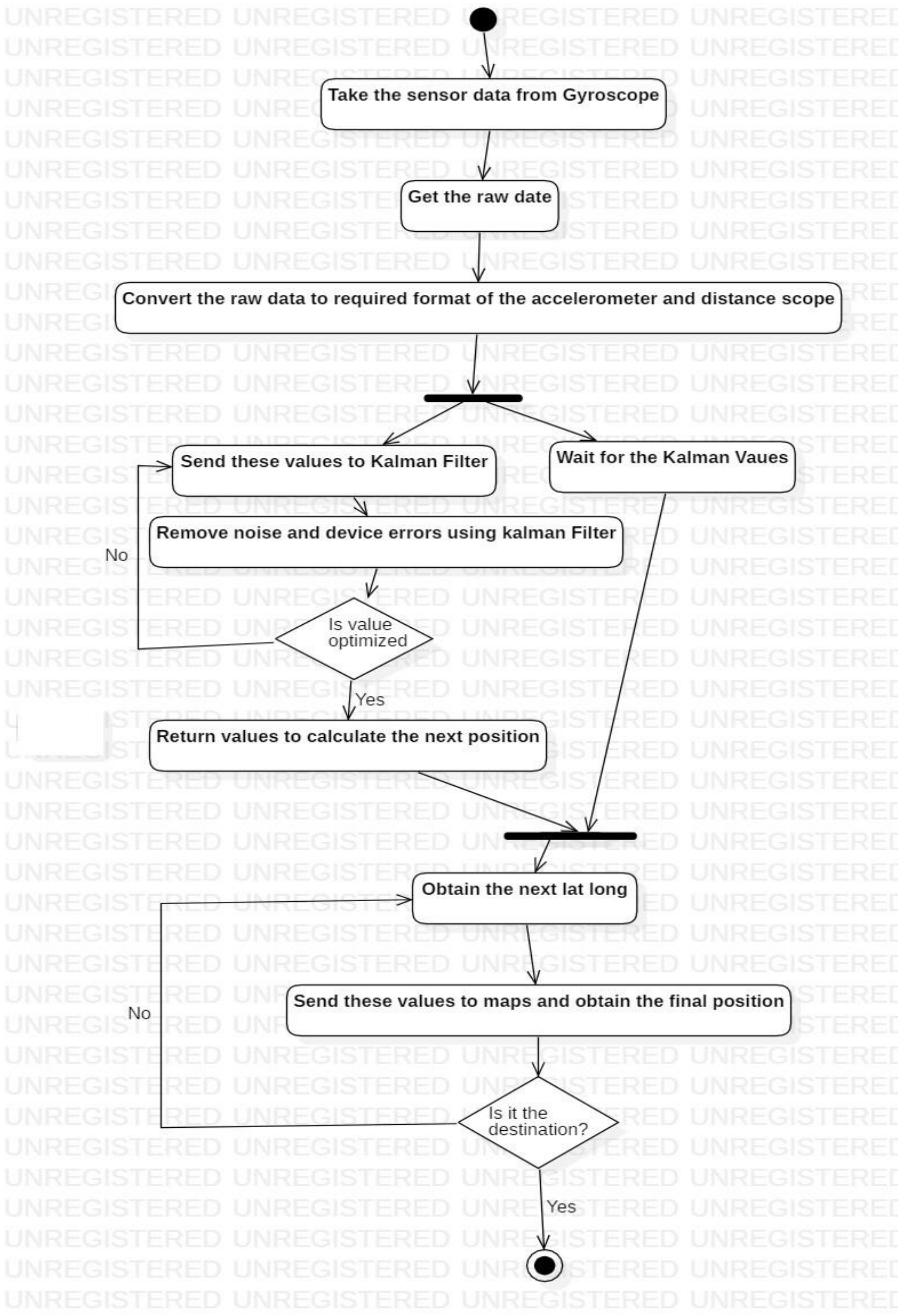

Take the sensor data from Gyroscope
Get the raw date
Convert the raw data to required format of the accelerometer and distance scope
Send these values to Kalman Filter
Wait for the Kalman Vaues
Remove noise and device errors using kalman Filter
No
Is value optimized
Yes
Return values to calculate the next position
Obtain the next lat long
No
Send these values to maps and obtain the final position
Is it the destination?
Yes

Fig 1. Activity diagram of Project

3.1.2.2 Flowchart:

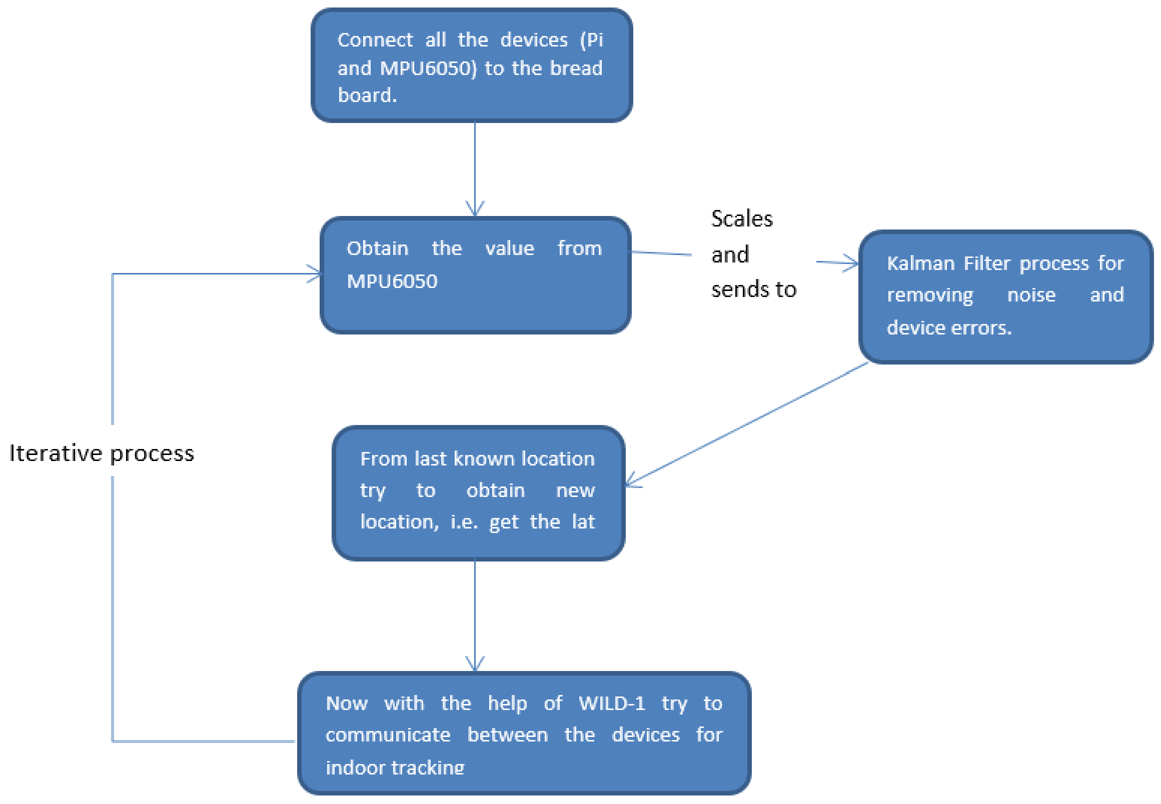

Connect all the devices (Pi and MPU6050) to the bread board.
Obtain the value from MPU6050
Scales and sends to
Kalman Filter process for removing noise and device errors.
From last known location try to obtain new location, i.e. get the lat
Now with the help of WILD-1 try to communicate between the devices for indoor tracking
Iterative process

Fig 2. Flowchart of the project

3.2 System Requirements

Our project requires the following requirements:

i. Hardware
ii. Software

i) Hardware Requirements:

a) Raspberry pi

It is an IoT gadget which establishes as an OS with variations in memory and processor ability. With different pins, it controls the outside gadgets associated with it. It has GPIO PINS, USB controller, Ethernet, Smaller scale USB SD card holder and so on. In this undertaking we incorporate this gadget with MPU6050 gadget and attempt to acquire the qualities from it and store it routinely. It is presently a current prevalent IoT gadget for all the coordinated gadgets.

Fig 3. Raspberry pi

b) *MPU6050:*

It is a 6-axis gadget utilized for distinguishing the increasing speed esteems and the direction esteems. It is internally coordinated with the accelerometer and the gyroscope. These two will give assistance to ascertain the location of the question which is the way to track a path. It is a precise and solid IMU sensor. It reports orientation, velocity magnetic waves and gravitational forces. Despite the fact that these are not acquired specifically from the gadget but rather can be gotten after couple of figuring and adjustments.

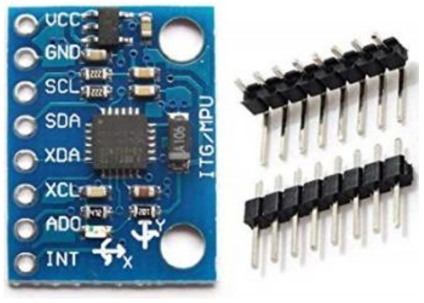


Fig 4. MPU 6050

c) *Connecting wires:*

Interfacing wires are utilized for associating pi and mpu6050 gadgets. We connect MPU6050 pins to the voltage, ground, in/out, serial clock, digital to analog pins of both devices.

Fig 5. Connecting Wires

Final connection:

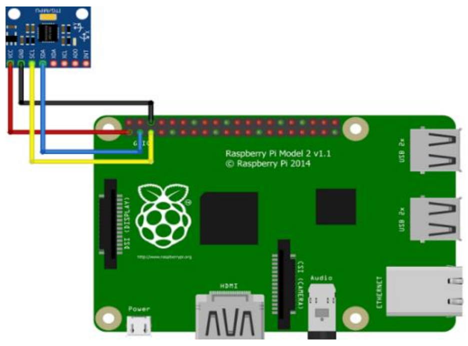


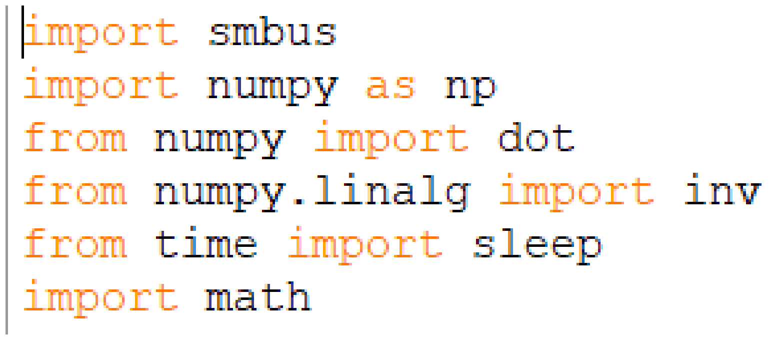
Fig 6. Final Connection

ii) Software Requirements

Python

CHAPTER 4

IMPLEMENTATION



```
import smbus
import numpy as np
from numpy import dot
from numpy.linalg import inv
from time import sleep
import math
```

Fig 7. Import Libraries

Import all the above libraries.

- smbus: This library is for interfacing the MPU6050 device with the raspberry Pi.

  The System Management Bus (abbreviated to SMBus or SMB) is a single-ended simple two-wire bus for the purpose of lightweight communication. Most commonly it is found in computer motherboards for communication with the power source for ON/OFF instructions.
  SMBus low power = 350 µA
  SMBus high power = 4 mA
  I²C-bus = 3 mA

NUMPY

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

- numpy: this library is for all mathematical calculations.

- numpy->dot: This function performs matrix calculations.
- numpy.linalg->inv: This function performs the inverse of a given matrix.

Python's time module has a handy function called sleep(). Essentially, as the name implies, it pauses your Python program. time.sleep() is the equivalent to the Bash shell's sleep command.
This is the syntax of the time.sleep() function: `time.sleep(secs),` secs - The number of seconds the Python program should pause execution. This argument should be either an int or a float.

- time->sleep: this is for giving a sleep time to the Kalman Filter.
- math: This library is used for various mathematical calculations as well.

Kalman Filter:

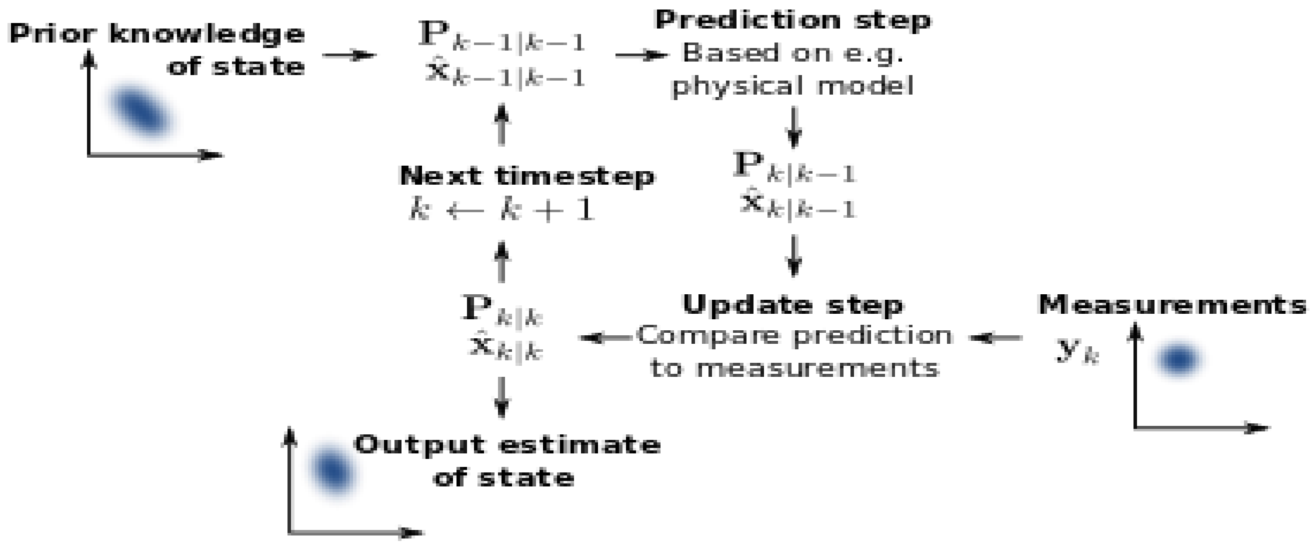
Kalman filtering, also known as linear quadratic estimation (LQE), is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each timeframe.

The algorithm works in a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (necessarily corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix; no additional past information is required.

Noisy sensor data, approximations in the equations that describe the system evolution, and external factors that are not accounted for all place limits on how well it is possible to determine the system's state. The Kalman filter deals effectively with the uncertainty due to noisy sensor data and to some extent also with random external factors. The Kalman filter produces an estimate of the state of the system as an average of the system's predicted state and of the new measurement using a weighted average. The weights are calculated from the covariance, a measure of the estimated uncertainty of the prediction of the system's state. The result of the weighted average is a new state estimate that lies between the predicted and measured state, and has a better estimated uncertainty than either alone.

This process is repeated at every time step, with the new estimate and its covariance informing the prediction used in the following iteration. This means that the Kalman filter works recursively and requires only the last "best guess", rather than the entire history, of a system's state to calculate a new state.

The relative certainty of the measurements and current state estimate is an important consideration, and it is common to discuss the response of the filter in terms of the Kalman filter's gain. The Kalman gain is the relative weight given to the measurements and current state estimate, and can be "tuned" to achieve particular performance. With a high gain, the filter places more weight on the most recent measurements, and thus follows them more responsively. With a low gain, the filter follows the model predictions more closely. At the extremes, a high gain close to one will result in a jumpier estimated trajectory, while low gain close to zero will smooth out noise but decrease the responsiveness.

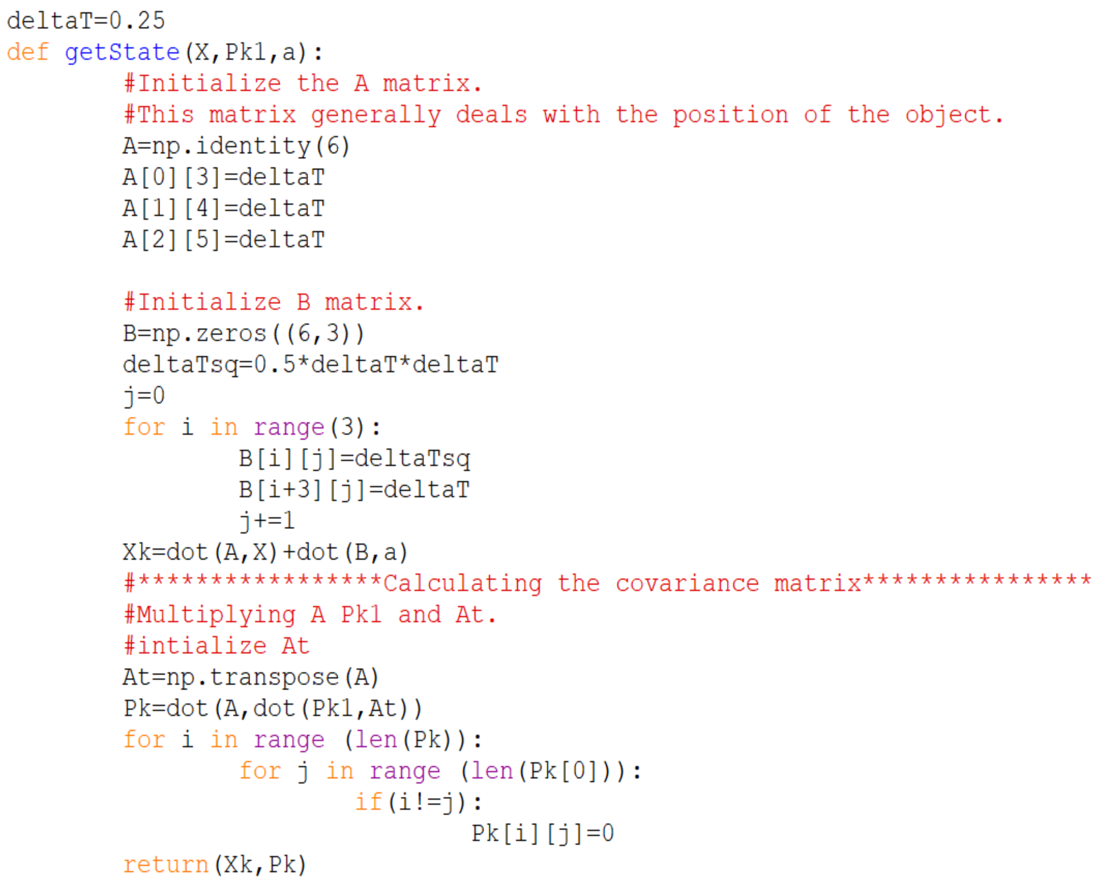


Prior knowledge of state
$\mathbf{P}_{k-1|k-1}$ $\hat{\mathbf{x}}_{k-1|k-1}$
Prediction step
Based on e.g. physical model
$\mathbf{P}_{k|k-1}$ $\hat{\mathbf{x}}_{k|k-1}$
Next timestep
$k \leftarrow k+1$
Update step
Compare prediction to measurements
Measurements
$\mathbf{y}_k$
$\mathbf{P}_{k|k}$ $\hat{\mathbf{x}}_{k|k}$
Output estimate of state

Fig 8. Flow of Kalman Filter

********************1. Kalman Filter Calculations******************

a. Get Initial State



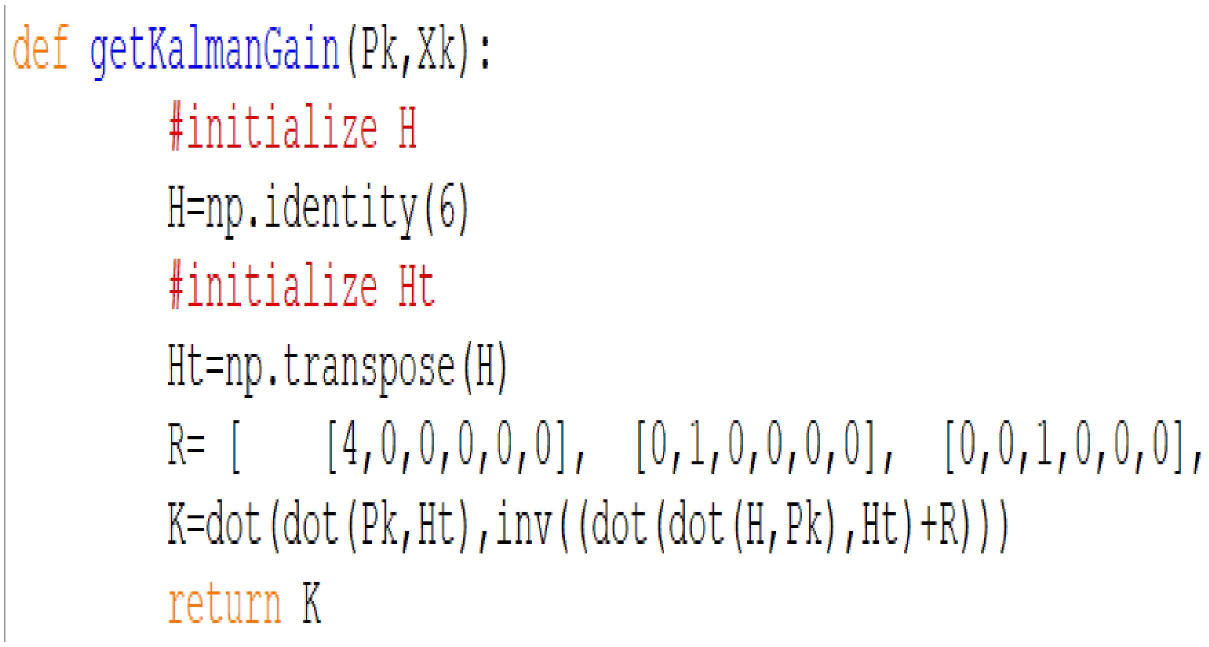
```
deltaT=0.25
def getState(X,Pk1,a):
        #Initialize the A matrix.
        #This matrix generally deals with the position of the object.
        A=np.identity(6)
        A[0][3]=deltaT
        A[1][4]=deltaT
        A[2][5]=deltaT

        #Initialize B matrix.
        B=np.zeros((6,3))
        deltaTsq=0.5*deltaT*deltaT
        j=0
        for i in range(3):
                B[i][j]=deltaTsq
                B[i+3][j]=deltaT
                j+=1
        Xk=dot(A,X)+dot(B,a)
        #****************Calculating the covariance matrix****************
        #Multiplying A Pk1 and At.
        #intialize At
        At=np.transpose(A)
        Pk=dot(A,dot(Pk1,At))
        for i in range (len(Pk)):
                for j in range (len(Pk[0])):
                        if(i!=j):
                                Pk[i][j]=0
        return(Xk,Pk)
```

Fig. 9 Kalman Filter Calculations

Here we do the following:

- First initialize the A matrix. Here, A is the matrix which catches the initial position of the object.
- Then initialize B matrix. This matrix will hold different time variants for calculating the position.
- Now, calculate the position using the Kalman filter formula $X_k' = AX_{k-1} + BU_k + W_k$ where U is control variable matrix, W is predicted state noise matrix. X_{k-1} is the previous state value.
- After this evaluate P_k value using the formula $P_k' = AP_{k-1}A^t + Q_k$, where Q is process noise covariance matrix (*This will keep the state covariance matrix from becoming too small or going to 0).*

a. Calculate Kalman Gain



```
def getKalmanGain(Pk,Xk):
        #initialize H
        H=np.identity(6)
        #initialize Ht
        Ht=np.transpose(H)
        R= [    [4,0,0,0,0,0],  [0,1,0,0,0,0],  [0,0,1,0,0,0],
        K=dot(dot(Pk,Ht),inv((dot(dot(H,Pk),Ht)+R)))
        return K
```

Fig 10. Calculate Kalman Gain

Here, we calculate the Kalman gain using the formula:

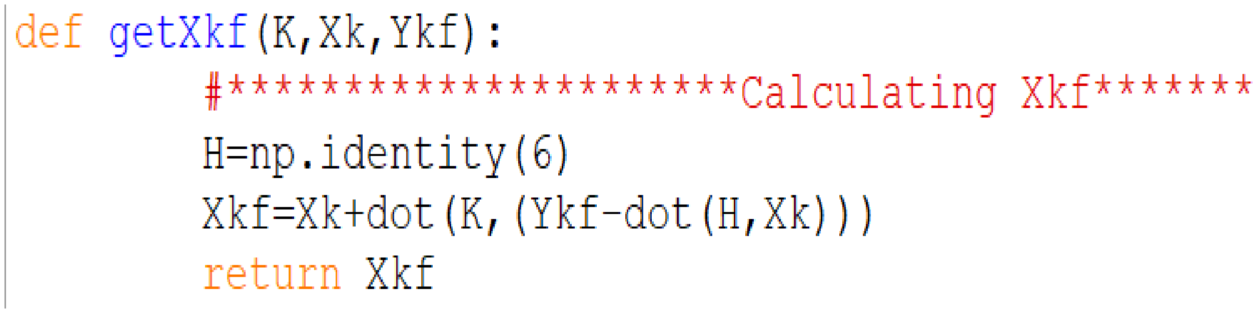
$$\boxed{K = \frac{P'_k H}{H P'_k H^T + R}}$$

Where, K is Kalman gain, R is sensor noise/measurement covariance matrix, H is conversion matrix (*to make sizes consistent*). For all these calculations we use the matrix transpose, multiplication and identity from the numpy library.

With this kalman gain we try to optimize the values obtained from the sensor.

This gain is useful for integrating the obtained sensor data.

b. Get state update



```
def getXkf(K,Xk,Ykf):
        #*********************Calculating Xkf*******
        H=np.identity(6)
        Xkf=Xk+dot(K,(Ykf-dot(H,Xk)))
        return Xkf
```

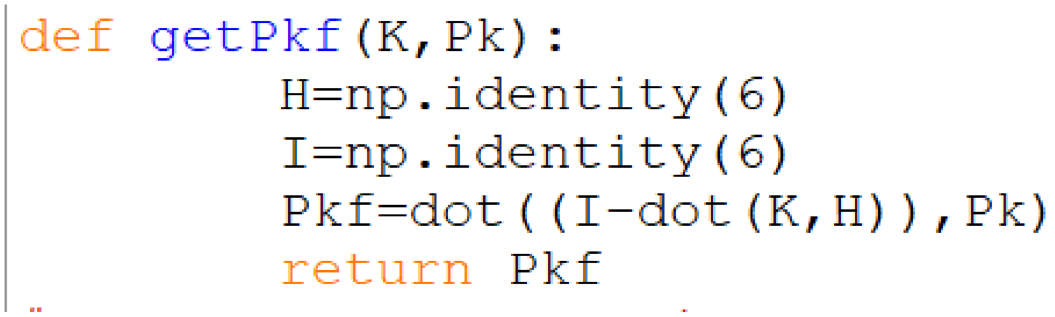
Fig 11. Get state update

In state updating process we try to obtain the new state using Kalman gain and measurement from sensor data.
Using the following formula we try to obtain the new state.

$$X_k = X'_k + K(Y_k - HX'_k)$$

Where, K is Kalman gain.

c. Get covariance update



```
def getPkf(K,Pk):
        H=np.identity(6)
        I=np.identity(6)
        Pkf=dot((I-dot(K,H)),Pk)
        return Pkf
```

Fig 12. Get covariance update

Update the covariance matrix using the following formula.

$$P_k = (I - KH)P'_k$$

Where H is measurement noise form the sensors. $P_k{}'$ is the previous covariance matrix and I is identity matrix.

**************************2. Reading sensor data**************************

- MPU6050 sensor module is an integrated 6-axis Motion tracking device.
- It has a 3-axis Gyroscope, 3-axis Accelerometer, Digital Motion Processor and a Temperature sensor, all in a single IC.
- It can accept inputs from other sensors like 3-axis magnetometer or pressure sensor using its Auxiliary I2C bus.
- If external 3-axis magnetometer is connected, it can provide complete 9-axis Motion Fusion output.
- A microcontroller can communicate with this module using I2C communication protocol. Various parameters can be found by reading values from addresses of certain registers using I2C communication.
- Gyroscope and accelerometer reading along X, Y and Z axes are available in 2's complement form.
- Gyroscope readings are in degrees per second (dps) unit; Accelerometer readings are in g unit.

Here, we will interface MPU6050 module with Raspberry Pi to read Gyroscope and Accelerometer value and print them.
We can interface MPU6050 module with Raspberry Pi using Python and C language. We will display the value of Accelerometer and Gyroscope on terminal which are read from MPU6050 module.

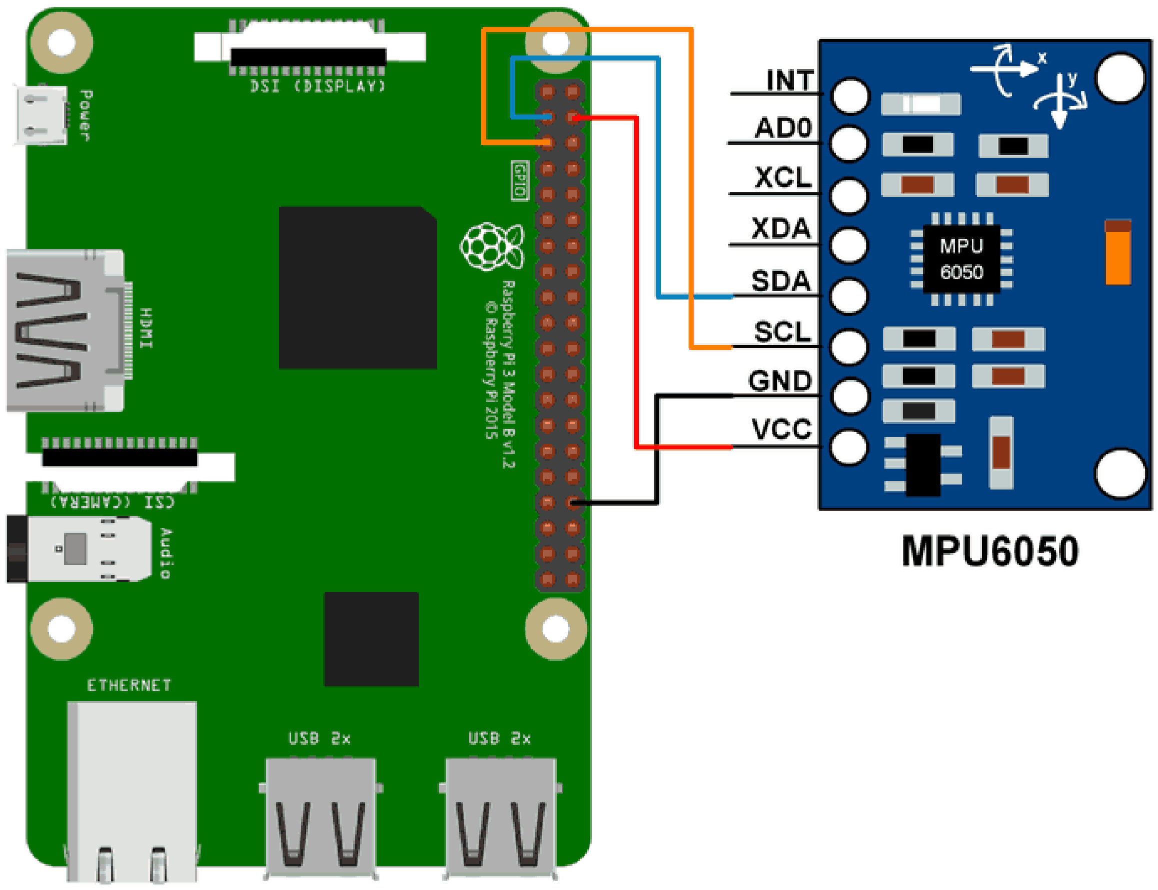


Fig 13. Raspberry Pi

```
#Powermanagement registers
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c

bus = smbus.SMBus(1)
address = 0x68

bus.write_byte_data(address, power_mgmt_1, 0)

def read_word(adr):
        high = bus.read_byte_data(address, adr)
        low = bus.read_byte_data(address, adr+1)
        val = (high << 8) + low
        return val

def read_word_2c(adr):
        val = read_word(adr)
        if(val >= 0x8000):
                return -((65535 - val) + 1)
        else:
                return val
```

Fig 14. Reading sensor data

Here, we have the following steps:

- Initialize the buses.
- Read the data values and store them in the registers.
- Get the accurate data if not up to the scale.

a. Initialize the buses

Assign the registers 0x6b and 0x6c for managing power registers.
Then using the library smbus get the smbus function and start interfacing with raspberry pi. Later get one register to store the data from the sensors.

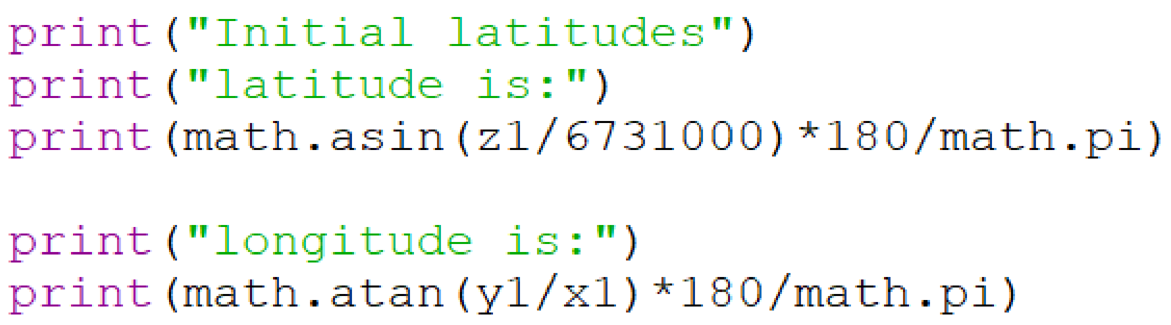
b. Read and store the data values

To read the data from the sensors, use the above functions and send them for scaling.

c. Get scaled data

Since registers are only 16 bit we have to scale the data according to the registers. For this scaling we use this scaling function.

**********************3.Calculating initial LatLong********************

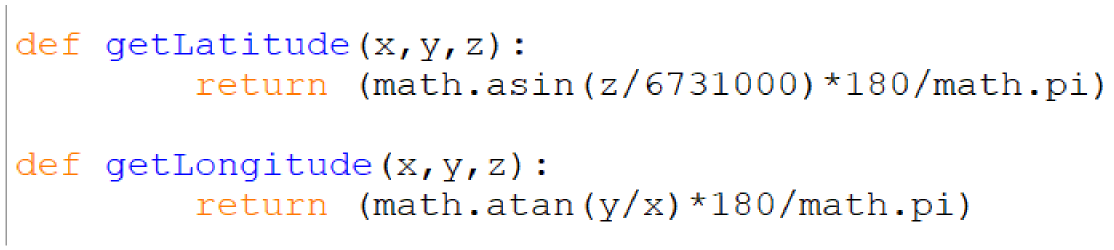


```
print("Initial latitudes")
print("latitude is:")
print(math.asin(z1/6731000)*180/math.pi)

print("longitude is:")
print(math.atan(y1/x1)*180/math.pi)
```

Fig 15. Calculating initial Lat Long

Using the above formulas calculate the initial latlong.

******************4. Create a method for calculating the LatLong*****************

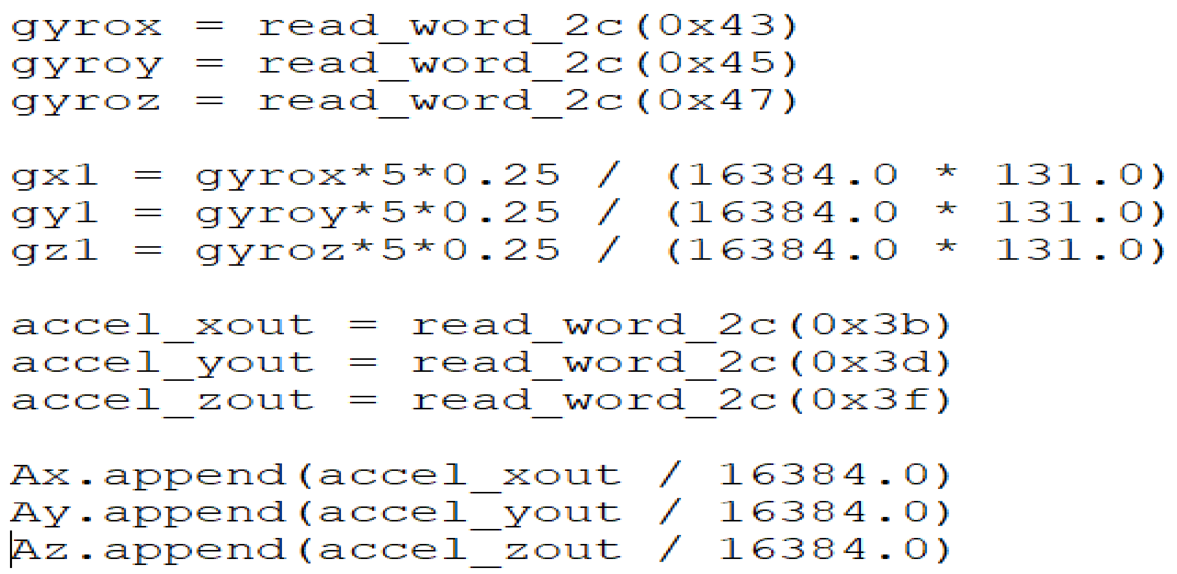


```
def getLatitude(x,y,z):
        return (math.asin(z/6731000)*180/math.pi)

def getLongitude(x,y,z):
        return (math.atan(y/x)*180/math.pi)
```

Fig 16. a method for calculating the Lat Long

These functions will generate the latlong when given x, y, z inclinations.

*******************5. Scale the values obtained from the sensor******************

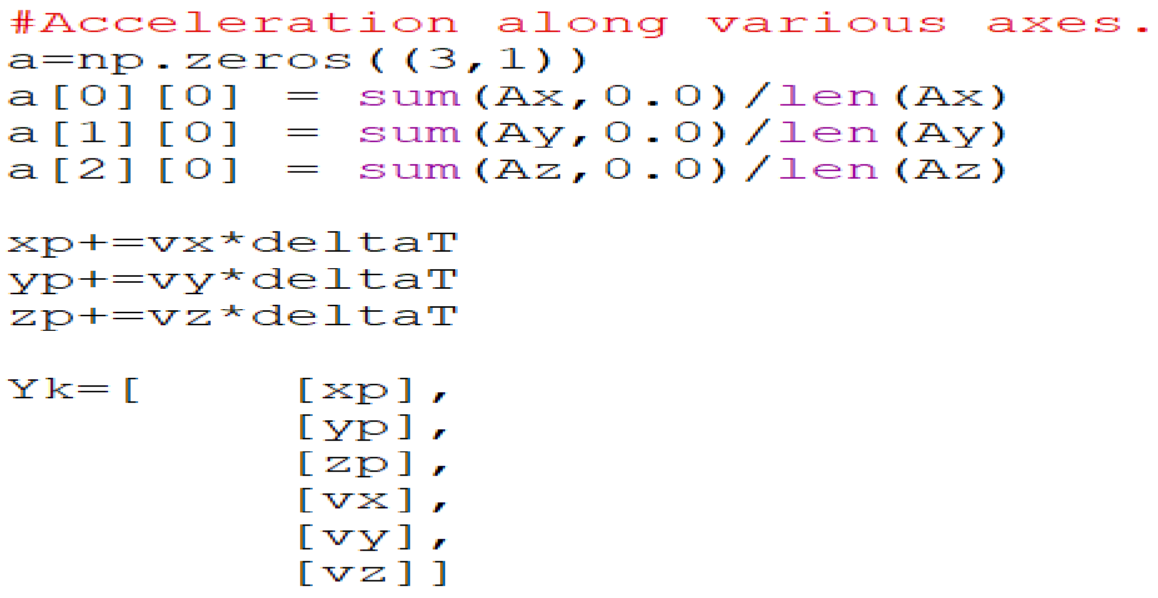


```
gyrox = read_word_2c(0x43)
gyroy = read_word_2c(0x45)
gyroz = read_word_2c(0x47)

gx1 = gyrox*5*0.25 / (16384.0 * 131.0)
gy1 = gyroy*5*0.25 / (16384.0 * 131.0)
gz1 = gyroz*5*0.25 / (16384.0 * 131.0)

accel_xout = read_word_2c(0x3b)
accel_yout = read_word_2c(0x3d)
accel_zout = read_word_2c(0x3f)

Ax.append(accel_xout / 16384.0)
Ay.append(accel_yout / 16384.0)
Az.append(accel_zout / 16384.0)
```

Fig 17. Scale the values obtained from the sensor

These are the crucial points of the Kalman filter. Here, we have to scale the data obtaining from the sensors. The sensors give the values which are not readable and calculable. Hence, we have to convert them to the requirement of the Kalman filter calculations.

*************************6. Calculating the Y_k matrix*************************

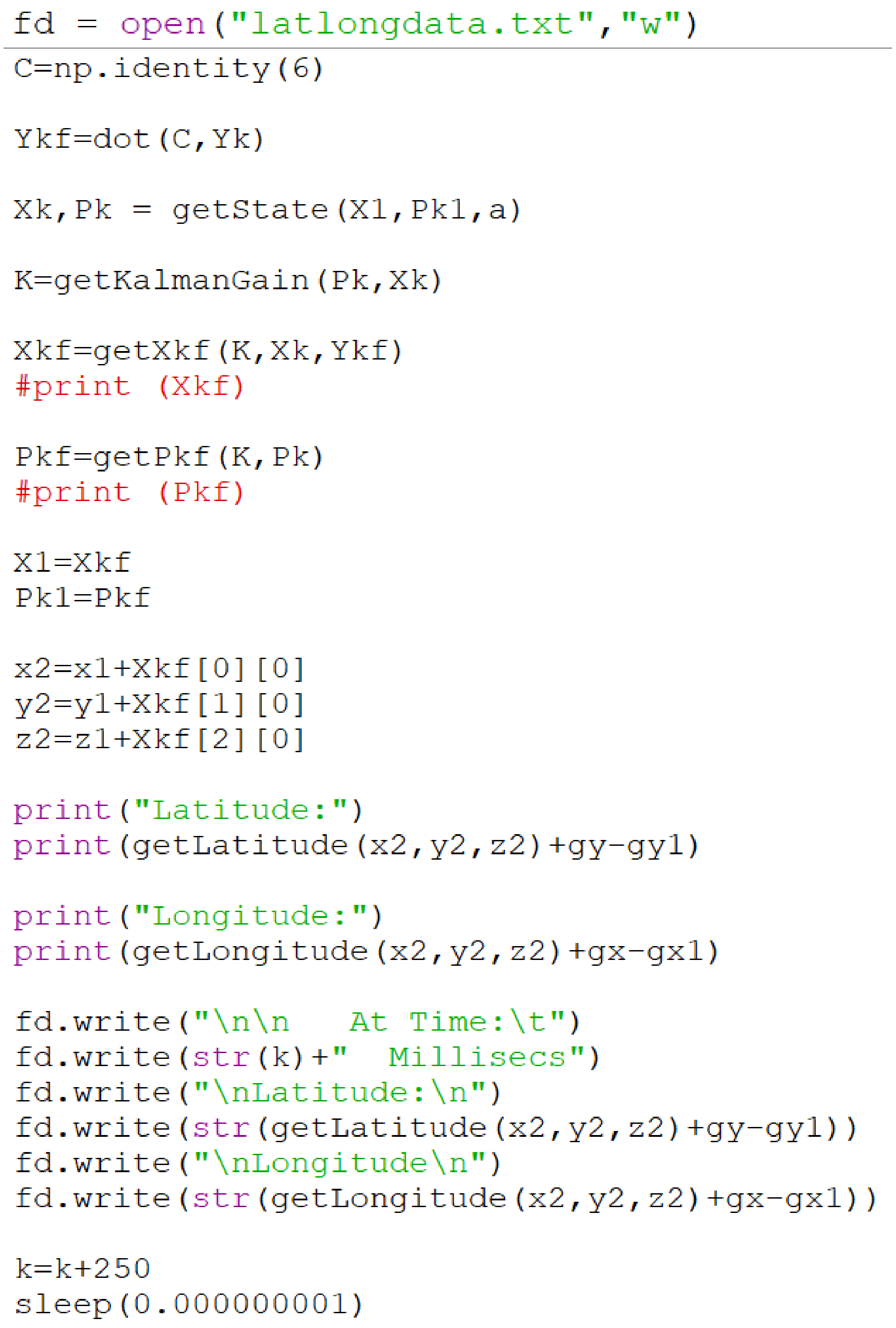


```
#Acceleration along various axes.
a=np.zeros((3,1))
a[0][0] = sum(Ax,0.0)/len(Ax)
a[1][0] = sum(Ay,0.0)/len(Ay)
a[2][0] = sum(Az,0.0)/len(Az)

xp+=vx*deltaT
yp+=vy*deltaT
zp+=vz*deltaT

Yk=[      [xp],
          [yp],
          [zp],
          [vx],
          [vy],
          [vz]]
```

Fig 18. Calculating the Y_k matrix

From the sensor get the distances after integrating the values twice and for velocity integrate it once.

**************************7.Mainfunction***********************



```
fd = open("latlongdata.txt","w")
C=np.identity(6)

Ykf=dot(C,Yk)

Xk,Pk = getState(X1,Pk1,a)

K=getKalmanGain(Pk,Xk)

Xkf=getXkf(K,Xk,Ykf)
#print (Xkf)

Pkf=getPkf(K,Pk)
#print (Pkf)

X1=Xkf
Pk1=Pkf

x2=x1+Xkf[0][0]
y2=y1+Xkf[1][0]
z2=z1+Xkf[2][0]

print("Latitude:")
print(getLatitude(x2,y2,z2)+gy-gy1)

print("Longitude:")
print(getLongitude(x2,y2,z2)+gx-gx1)

fd.write("\n\n   At Time:\t")
fd.write(str(k)+"  Millisecs")
fd.write("\nLatitude:\n")
fd.write(str(getLatitude(x2,y2,z2)+gy-gy1))
fd.write("\nLongitude\n")
fd.write(str(getLongitude(x2,y2,z2)+gx-gx1))

k=k+250
sleep(0.000000001)
```

Fig 19. Main Function

Now, from the above code we can see that we are doing the initial calculations before calling the Kalman filter function. This is a necessary step and for formatting the values. After calling Kalman filter function, we try to obtain the latlong of evaluated data. This then is stored in a file latlongdata.txt.

********************************8.Plotting********************************

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

matplotlib.pyplot.acorr(*x*, *hold=None*, *data=None*, ***kwargs*)

Plot the autocorrelation of x.

Parameters:	**x** : sequence of scalar
	hold : boolean, optional, *deprecated*, default: True
	detrend : callable, optional, default: `mlab.detrend_none`
	x is detrended by the `detrend` callable. Default is no normalization.
	normed : boolean, optional, default: True
	if True, input vectors are normalised to unit length.
	usevlines : boolean, optional, default: True

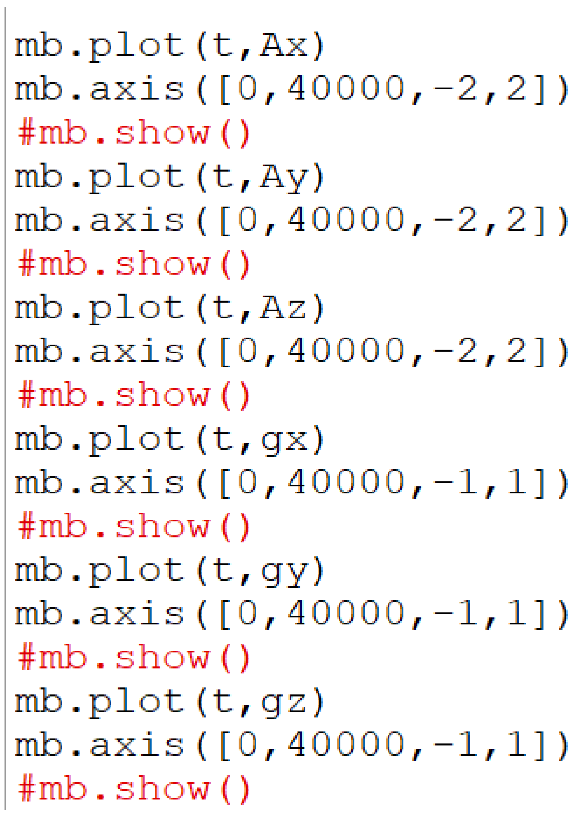
if True, Axes.vlines is used to plot the vertical lines from the origin to the acorr. Otherwise, Axes.plot is used.

maxlags : integer, optional, default: 10

number of lags to show. If None, will return all 2 * len(x) - 1 lags.

Returns: **(lags, c, line, b)** : where:

- `lags` are a length 2`maxlags+1 lag vector.
- `c` is the 2`maxlags+1 auto correlation vectorI
- `line` is a `Line2D`instance returned by `plot`.
- `b` is the x-axis.



```
mb.plot(t,Ax)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,Ay)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,Az)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,gx)
mb.axis([0,40000,-1,1])
#mb.show()
mb.plot(t,gy)
mb.axis([0,40000,-1,1])
#mb.show()
mb.plot(t,gz)
mb.axis([0,40000,-1,1])
#mb.show()
```

Fig 20. Plotting1

***********************9. Graphing***********************

numpy.polyfit(*x*, *y*, *deg*, *rcond=None*, *full=False*, *w=None*, *cov=False***)**

Least squares polynomial fit.

Fit a polynomial `p(x) = p[0] * x**deg + ... + p[deg]` of degree *deg* to points *(x, y)*. Returns a vector of coefficients *p* that minimises the squared error in the order *deg*, *deg-1*, … *0*.

The **Polynomial.fit** class method is recommended for new code as it is more stable numerically. See the documentation of the method for more information.

PARAMETERS:

x : *array_like, shape (M,)*

x-coordinates of the M sample points `(x[i], y[i])`.

y : *array_like, shape (M,) or (M, K)*

y-coordinates of the sample points. Several data sets of sample points sharing the same x-coordinates can be fitted at once by passing in a 2D-array that contains one dataset per column.

deg : *int*

Degree of the fitting polynomial

rcond : *float, optional*

Relative condition number of the fit. Singular values smaller than this relative to the largest singular value will be ignored. The default value is len(x)*eps, where eps is the relative precision of the float type, about 2e-16 in most cases.

full : *bool, optional*

Switch determining nature of return value. When it is False (the default) just the coefficients are returned, when True diagnostic information from the singular value decomposition is also returned.

w : *array_like, shape (M,),optional*

Weights to apply to the y-coordinates of the sample points. For gaussian uncertainties, use 1/sigma (not 1/sigma**2).

cov : *bool or str, optional*

If given and not *False*, return not just the estimate but also its covariance matrix. By default, the covariance are scaled by chi2/sqrt(N-dof), i.e., the weights are presumed to be unreliable except in a relative sense and everything is scaled such that the reduced chi2 is unity. This scaling is omitted if `cov='unscaled'`, as is relevant for the case that the weights are 1/sigma**2, with sigma known to be a reliable estimate of the uncertainty.

RETURNS:

p : *ndarray, shape (deg + 1,) or (deg + 1, K)*

Polynomial coefficients, highest power first. If *y* was 2-D, the coefficients for *k*-th data set are in `p[:,k]`.

residuals, rank, singular_values, rcond

Present only if **full** = True. Residuals of the least-squares fit, the effective rank of the scaled Vandermonde coefficient matrix, its singular values, and the specified value of *rcond*. For more details, see **linalg.lstsq**.

V : *ndarray, shape (M,M) or (M,M,K)*

Present only if **full** = False and *cov`=True. The covariance matrix of the polynomial coefficient estimates. The diagonal of this matrix are the variance estimates for each coefficient. If y is a 2-D array, then the covariance matrix for the `k*-th data set are in `V[:,:,k]`

RankWarning

The rank of the coefficient matrix in the least-squares fit is deficient. The warning is only raised if **full** = False.

The warnings can be turned off by

```
>>> import warnings
>>> warnings.simplefilter('ignore', np.RankWarn
ing)
```

class **numpy.poly1d(***c_or_r*, *r=False*, *variable=None***)[source]**

A one-dimensional polynomial class.

A convenience class, used to encapsulate “natural” operations on polynomials so that said operations may take on their customary form in code (see Examples).

PARAMETERS:

c_or_r : *array_like*

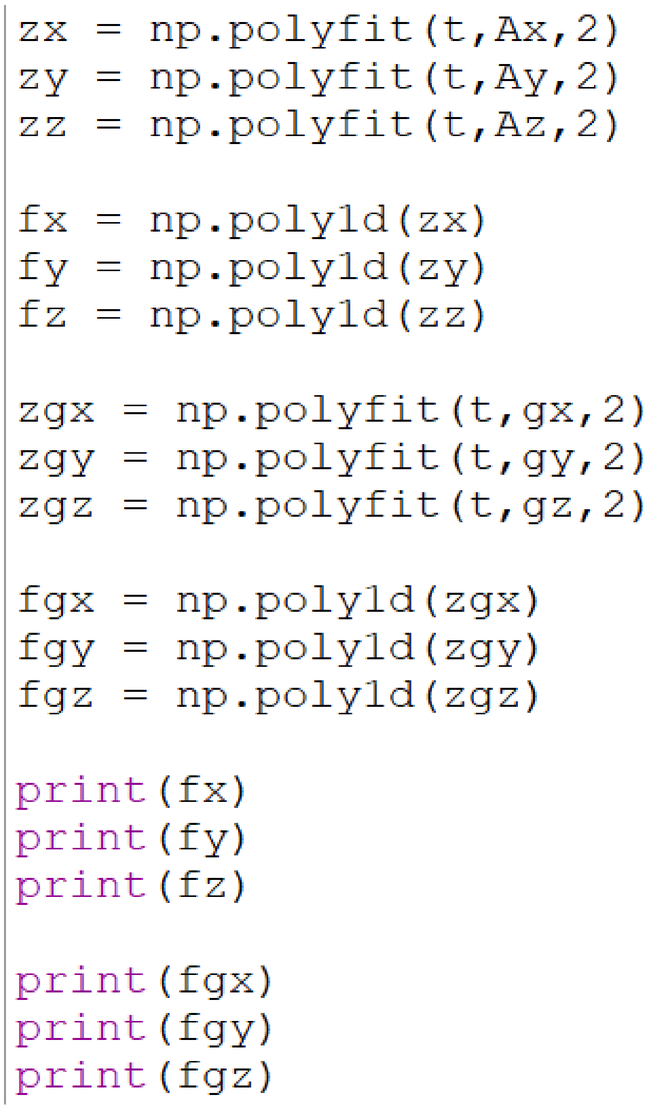
The polynomial’s coefficients, in decreasing powers, or if the value of the second parameter is True, the polynomial’s roots (values where the polynomial evaluates to 0). For example, `poly1d([1, 2, 3])` returns an object that represents , whereas `poly1d([1, 2, 3], True)` returns one that represents .

r : *bool, optional*

If True, *c_or_r* specifies the polynomial’s roots; the default is False.

variable : *str, optional*

Changes the variable used when printing *p* from *x* to **variable** (see Examples).



```
zx = np.polyfit(t,Ax,2)
zy = np.polyfit(t,Ay,2)
zz = np.polyfit(t,Az,2)

fx = np.poly1d(zx)
fy = np.poly1d(zy)
fz = np.poly1d(zz)

zgx = np.polyfit(t,gx,2)
zgy = np.polyfit(t,gy,2)
zgz = np.polyfit(t,gz,2)

fgx = np.poly1d(zgx)
fgy = np.poly1d(zgy)
fgz = np.poly1d(zgz)

print(fx)
print(fy)
print(fz)

print(fgx)
print(fgy)
print(fgz)
```

Fig 21. Plotting2

*************************10. Polval and filing data*************************

numpy.polyval(*p*, *x*)

Evaluate a polynomial at specific values.

If *p* is of length N, this function returns the value:

`p[0]*x**(N-1) + p[1]*x**(N-2) + ... + p[N-2]*x + p[N-1]`

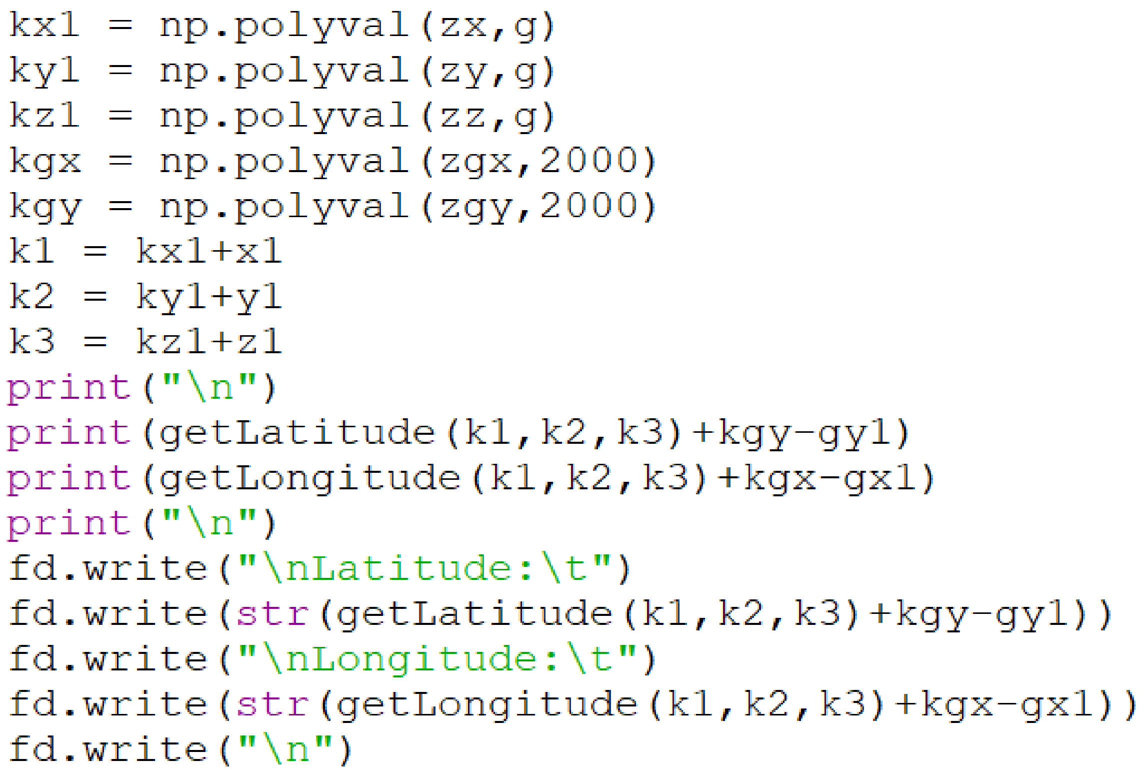
If *x* is a sequence, then *p(x)* is returned for each element of *x*. If *x* is another polynomial then the composite polynomial *p(x(t))* is returned.

PARAMETERS:

p *: array_like or poly1d object*
1D array of polynomial coefficients (including coefficients equal to zero) from highest degree to the constant term, or an instance of poly1d.
x *: array_like or poly1d object*
A number, an array of numbers, or an instance of poly1d, at which to evaluate *p*.
values *: ndarray or poly1d*

RETURNS:

If *x* is a poly1d instance, the result is the composition of the two polynomials, i.e., *x* is “substituted” in *p* and the simplified result is returned. In addition, the type of *x* - array_like or poly1d - governs the type of the output: *x*array_like => *values* array_like, *x* a poly1d object => *values* is also.



```
kx1 = np.polyval(zx,g)
ky1 = np.polyval(zy,g)
kz1 = np.polyval(zz,g)
kgx = np.polyval(zgx,2000)
kgy = np.polyval(zgy,2000)
k1 = kx1+x1
k2 = ky1+y1
k3 = kz1+z1
print("\n")
print(getLatitude(k1,k2,k3)+kgy-gy1)
print(getLongitude(k1,k2,k3)+kgx-gx1)
print("\n")
fd.write("\nLatitude:\t")
fd.write(str(getLatitude(k1,k2,k3)+kgy-gy1))
fd.write("\nLongitude:\t")
fd.write(str(getLongitude(k1,k2,k3)+kgx-gx1))
fd.write("\n")
```

Fig 22. Polval and filing data

CHAPTER 5

RESULTS AND DISCUSSIONS

RASPBERRY PI platform is most used after ADRUINO. Although overall applications of PI are less it is most preferred when developing advanced applications. Also, the RASPBERRY PI is an open source platform where one can get a lot of related information so you can customize the system depending on the need.

The MPU6050 is a Micro Electro-Mechanical Systems (MEMS) which consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion related parameter of a system or object. This module also has a (DMP) Digital Motion Processor inside it which is powerful enough to perform complex calculation and thus free up the work for Microcontroller.

Latitude:	17.3918252814
Longitude:	78.322087892
Latitude:	17.3918253058
Longitude:	78.3220878085
Latitude:	17.3918253287
Longitude:	78.3220877319
Latitude:	17.3918253499
Longitude:	78.3220876623
Latitude:	17.3918253696
Longitude:	78.3220875995

Latitude:	17.3918253877
Longitude:	78.3220875437
Latitude:	17.3918254042
Longitude:	78.3220874947
Latitude:	17.3918254191
Longitude:	78.3220874527
Latitude:	17.3918254325
Longitude:	78.3220874176
Latitude:	17.3918254443
Longitude:	78.3220873894
Latitude:	17.3918254545
Longitude:	78.3220873682
Latitude:	17.3918254632
Longitude:	78.3220873538
Latitude:	17.3918254702
Longitude:	78.3220873463
Latitude:	17.3918254757
Longitude:	78.3220873458
Latitude:	17.3918254797
Longitude:	78.3220873522

```
Latitude:    17.391825482
Longitude: 78.3220873655

Latitude:    17.3918254828
Longitude: 78.3220873857

Latitude:    17.391825482
Longitude: 78.3220874128

Latitude:    17.3918254796
Longitude: 78.3220874468

Latitude:    17.3918254756
Longitude: 78.3220874877
```

Fig 23. Output

The above pictures will tell the lat longs of the output device. These Lat longs are later sent to the base map to obtain the location of the current position. This base map is where the object is seen moving along. All the points marked here have a particular name and these are displayed if the object is hovered around in those places.

The combination of these two components specifies the position of any location on the surface of Earth, without consideration of altitude or depth. The grid formed by lines of latitude and longitude is known as a "graticule". The origin/zero point of this system is located in the Gulf of Guinea about 625 km (390 mi) south of Tema, Ghana.

- EKF are needed when you either have a non-linear process or measurement relationship
- Uses function f for difference equation and function h for the measurement equation
- No longer use optimal estimator (only in linear cases)
- Considered by some as the de facto standard for non-linear state estimation
- Heavily used in navigation systems and GPS

The Kalman Filter is a good and easy to use filter to get more reliable output from your sensors. The recursive approach makes it usable for real-time purposes such as in robots But, you have to be able to describe the underlying model properly.

4.1 Comparison to other filters

4.1.1 Particle Filter or Sequential Monte Carlo methods

- Estimate density represented with particles ⇒ Multimodal
- Does not require Gaussian noise
- Often used in complex non-linear models
- Large state space dimensionality requires lots of particles

4.1.2 Hybrid

- Particle Filter superior for dealing with multi-modal data
- EKF superior for dealing with updates with little noise
- Use PF until variance is below a certain level, switch to KF

Map projection formulas depend in the geometry of the projection as well as parameters dependent on the particular location at which the map is projected. The set of parameters can vary based on type of project and the conventions chosen for the projection.

Here are few examples where RASPBERRY PI 3 is chosen over other microcontrollers and development boards:

1. Where the system processing is huge. Most ARDUINO boards all have clock speed of less than 100MHz, so they can perform functions limited to their capabilities. They cannot process high end programs for applications like Weather Station, Cloud server, gaming console etc. With **1.2GHz clock speed** and **1 GB RAM RASPBERRY PI** can perform all those advanced functions.

2. Where wireless connectivity is needed. RASPBERRY PI 3 has wireless LAN and Bluetooth facility by which you can setup WIFI HOTSPOT for internet connectivity. For **Internet of Things** this feature is best suited.

3. RASPBERRY PI had dedicated port for connecting touch LCD display which is a feature that completely omits the need of monitor.

4. RASPBERRY PI also has dedicated camera port so one can connect camera without any hassle to the PI board.

5. RASPBERRY PI also has PWM outputs for application use.

There are many other features like HD steaming which further promote the use of RASPBERRY PI.

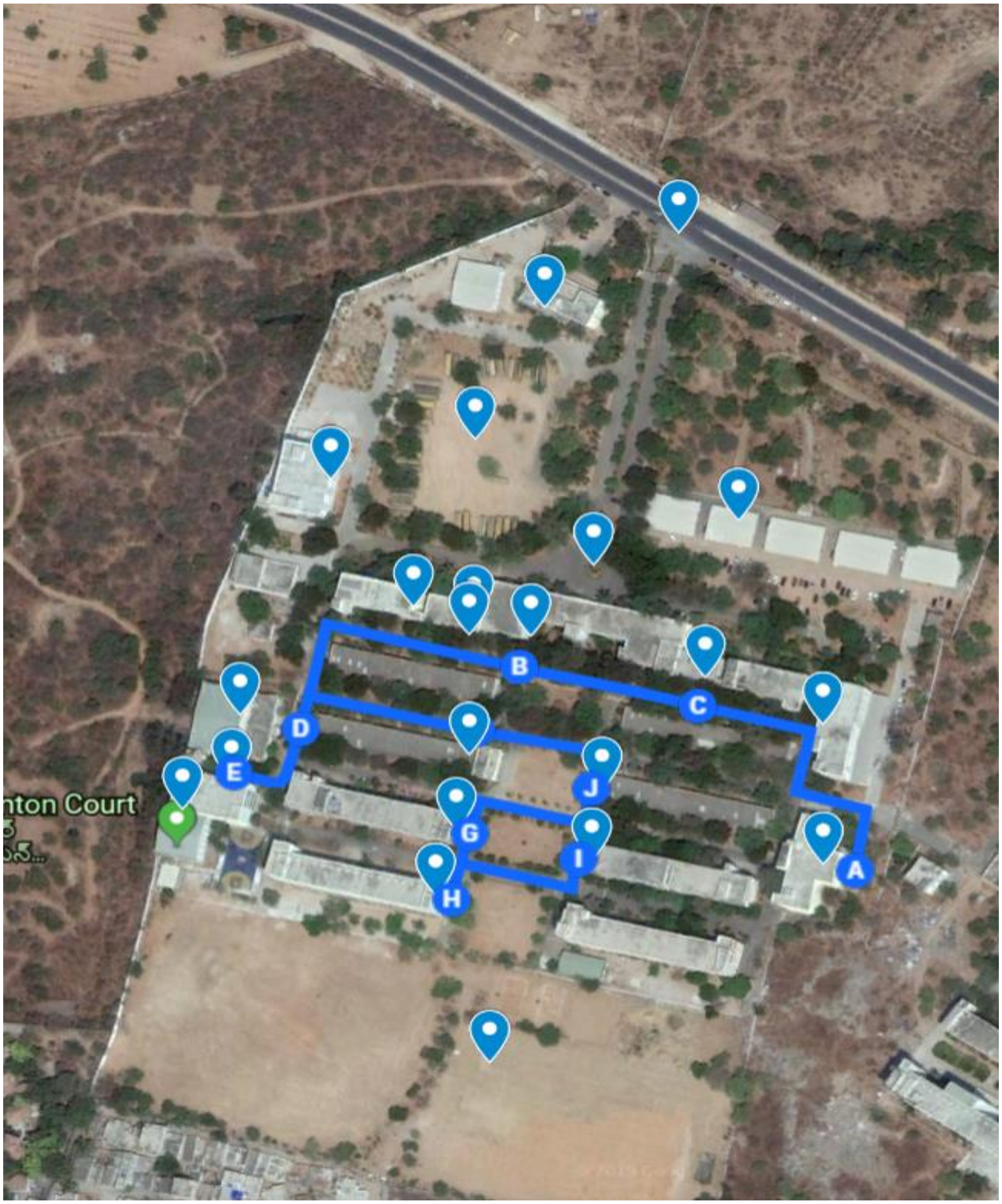


Fig 24: Base map

CHAPTER 6

CONCLUSION AND FUTURE WORK

Conclusion:

Our project titled Target Tracking System Using Kalman Filter is performed and the results are computed. Kalman filter provides 90%efficeint output even in the noisy environment. In this thesis we have studied several estimation and data association methods for target tracking. A general method of increasing the sampling frequency of a vision sensor by using a predictive Kalman filter and partial window imaging has been introduced and has been demonstrated to work effectively. The method reduces the acquisition and processing time of an image. The acquisition time is reduced by a larger percentage than the processing time and so the image processing is the bottle neck in reducing the sampling frequency. Two processing methods were implemented. This system was not as robust but it does provide a further increase in sampling frequency.

Future Scope:

This can further be used for various other applications. This system van only shows the result for 5km radius. For more extensions we require to optimize it.

REFERENCES

[1] Rameshbabu, J.Swarnadurga, G.Archana, K.Menaka., International Journal of Advanced Engineering Research and Studies: OBJECT TRACKING SYSTEM USING KALMAN FILTER Research Paper, Oct.-Dec.,2012

[2] Hua Yang(Univ. of North Carolina at Chapel Hill)G. Welch, Model-Based 3D Object Tracking Using an Extended-Extended Kalman Filter and Graphics Rendered Measurements IEEE paper24 April 2006.

[3] Jerel Nielsen, Randal Beard; Relative target estimation using a cascade of extended Kalman filters; Student publications, 2017-09-19.

[4] Dinesh M.A., Santhosh Kumar S., Sanath J. Shetty, Akarsh K.N., Manoj Gowda K.M.; Development of an Autonomous Drone for Surveillance Application; Research Journal 08 Sep, 2018

[5] United States Patent Application Publication Pub. No.: US 2018 / 0023953 A1 Roumeliotis et al Pub. Date: Jan. 25, 2018

[6] Daniel P. Koch,David O. Wheeler; Relative Multiplicative Extended Kalman Filter for Observable GPS-Denied Navigation; Faculty Publications; 2017-08-18.

[7] Donald Reid. An algorithm for tracking multiple targets. IEEE transactions on Automatic Control, 24(6):843–854, 1979. Published in the proceedings of ION GNSS+ 2017.

[8] B-NVoandW-Kma The gaussian mixture probability hypothesis density filter. IEEE Transactions on signal processing, 54(11):4091–4104, 2006.

[9] Peter C Niedfeldt. Recursive-RANSAC: A novel algorithm for tracking multiple targets in clutter. PhD thesis, 2014.

[10] Peter C Niedfeldt, Kyle Ingersoll, and Randal W Beard. Comparison and analysis of recursive-ransac for multiple target tracking. IEEE Transactions on Aerospace and Electronic Systems, 53(1):461–476, 2017.

[11] Peter C Niedfeldt and Randal W Beard. Convergence and complexity analysis of recursive-ransac: a new multiple target tracking algorithm. IEEE Transactions on Automatic Control, 61(2):456–461, 2016.

[12] Kyle Ingersoll, Peter C Niedfeldt, and Randal W Beard. Multiple target tracking and stationary object detection in video with recursive-ransac and tracker-sensor feedback. In Unmanned Aircraft Systems (ICUAS), 2015 International Conference on, pages 1320–1329. IEEE, 2015.

[13] Havran, V., Herzog, R., Seidel, H.-P.: On Fast Construction of Spatial Hierarchies for Ray Tracing. In: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, pp. 71–80 (September 2006)

[14] Benthin, C.: Realtime Raytracing on Current CPU Architectures. PhD thesis, Saarland University (2006)

[15] Popov, S., Gunther, J., Seidel, H.-P., Slusallek, P.: Experiences with Streaming Construction of SAH KDTrees. In: Proceedings of IEEE Symposium on Interactive Ray Tracing, pp. 89–94 (September 2006)

[16] Cleary, J.G., Wyvill, G.: Analysis Of An Algorithm For Fast Ray Tracing Using Uniform Space Subdivision. The Visual Computer (4), 65–83 (1988).

APPENDIX 1

CODE:

```
import matplotlib.pyplot as mb
import smbus
import numpy as np
from numpy import dot
from numpy.linalg import inv
from time import sleep
import math
""
#Prompt for time interval.
#deltaT=float(input("Enter the time interval:\t"));
deltaT=0.25
def getState(X,Pk1,a):
	#Initialize the A matrix.
	#This matrix generally deals with the position of the object.
	A=np.identity(6)
	A[0][3]=deltaT
	A[1][4]=deltaT
	A[2][5]=deltaT

	#Initialize B matrix.
	B=np.zeros((6,3))
	deltaTsq=0.5*deltaT*deltaT
	j=0
	for i in range(3):
		B[i][j]=deltaTsq
		B[i+3][j]=deltaT
		j+=1
	Xk=dot(A,X)+dot(B,a)
	#*****************Calculating	the	covariance
matrix****************
	#Multiplying A Pk1 and At.
	#intialize At
	At=np.transpose(A)
	Pk=dot(A,dot(Pk1,At))
	for i in range (len(Pk)):
		for j in range (len(Pk[0])):
			if(i!=j):
				Pk[i][j]=0
	return(Xk,Pk)
def getKalmanGain(Pk,Xk):
	#initialize H
	H=np.identity(6)
	#initialize Ht
	Ht=np.transpose(H)
	R= [	[4,0,0,0,0,0],		[0,1,0,0,0,0],		[0,0,1,0,0,0],
	[0,0,0,4,0,0],		[0,0,0,0,1,0],		[0,0,0,0,0,1]]
	K=dot(dot(Pk,Ht),inv((dot(dot(H,Pk),Ht)+R)))
	return K
def getXkf(K,Xk,Ykf):
	#**********************Calculating
Xkf************************
	H=np.identity(6)
	Xkf=Xk+dot(K,(Ykf-dot(H,Xk)))
	return Xkf
def getPkf(K,Pk):
	H=np.identity(6)
	I=np.identity(6)
	Pkf=dot((I-dot(K,H)),Pk)
	return Pkf
"""
#Powermanagement registers
power_mgmt_1 = 0x6b
power_mgmt_2 = 0x6c
```

```
bus = smbus.SMBus(1)
address = 0x68

bus.write_byte_data(address, power_mgmt_1, 0)

def read_word(adr):
        high = bus.read_byte_data(address, adr)
        low = bus.read_byte_data(address, adr+1)
        val = (high << 8) + low
        return val

def read_word_2c(adr):
        val = read_word(adr)
        if(val >= 0x8000):
                return -((65535 - val) + 1)
        else:
                return val
"""
xp=0
yp=0
zp=0

X1=np.zeros((6,1))
"""
x1=6731000*math.cos(17.466439*math.pi/180)*math.cos(72.489528
*math.pi/180)
y1=6731000*math.cos(17.466439*math.pi/180)*math.sin(72.489528
*math.pi/180)
z1=6731000*math.sin(17.466439*math.pi/180)
"""
X1[0][0] = x1
X1[1][0] = y1
X1[2][0] = z1
```

```
#initialize Pk1
Pk1 = [        [4,0,0,0,0,0],          [0,4,0,0,0,0],          [0,0,4,0,0,0],
        [0,0,0,4,0,0],          [0,0,0,0,4,0],          [0,0,0,0,0,4]]

print("Initial latitudes")
print("latitude is:")
print(math.asin(z1/6731000)*180/math.pi)

print("longitude is:")
print(math.atan(y1/x1)*180/math.pi)
"""
def getLatitude(x,y,z):
        return (math.asin(z/6731000)*180/math.pi)

def getLongitude(x,y,z):
        return (math.atan(y/x)*180/math.pi)

fd = open("latlongdata.txt","w")
gyrox = read_word_2c(0x43)
gyroy = read_word_2c(0x45)
gyroz = read_word_2c(0x47)

gx1 = gyrox*5*0.25 / (16384.0 * 131.0)
gy1 = gyroy*5*0.25 / (16384.0 * 131.0)
gz1 = gyroz*5*0.25 / (16384.0 * 131.0)

fd = open("data1.txt","w")
count=0
i=0
Ax=[]
Ay=[]
Az=[]

gx=[]
```

```
gy=[]
gz=[]

t=[]
k=0
while(count!=5000):

        accel_xout = read_word_2c(0x3b)
        accel_yout = read_word_2c(0x3d)
        accel_zout = read_word_2c(0x3f)

        Ax.append(accel_xout / 16384.0)
        Ay.append(accel_yout / 16384.0)
        Az.append(accel_zout / 16384.0)

        vx = np.trapz(Ax,dx=0.001)
        vy = np.trapz(Ay,dx=0.001)
        vz = np.trapz(Az,dx=0.001)

        gyrox = read_word_2c(0x43)
        gyroy = read_word_2c(0x45)
        gyroz = read_word_2c(0x47)

        gx.append(gyrox*3.3*0.25 / (16384.0 * 131.0))
        gy.append(gyroy*3.3*0.25 / (16384.0 * 131.0))
        gz.append(gyroz*3.3*0.25 / (16384.0 * 131.0))

        t.append(count)
        """
        print("\nAt time : ")
        print(count)
        print("Millisecs\n"+"\n Accelerometer")
        print(Ax[i])
        print(Ay[i])
        print(Az[i])
        print("Gyro")
        print(gx[i])
        print(gy[i])
        print(gz[i])

        fd.write("\n\n   At Time:\t")
        fd.write(str(count)+"  Millisecs")
        fd.write("\nAccelerometer\n"+str(Ax[i]))
        fd.write("\n"+str(Ay[i]))
        fd.write("\n"+str(Az[i]))
        fd.write("\nGyro:\n"+str(gx[i]))
        fd.write("\n"+str(gy[i]))
        fd.write("\n"+str(gz[i]))
        """
        count = count+250
        sleep(.25)
        i=i+1
        """
        #Acceleration along various axes.
        a=np.zeros((3,1))
        a[0][0] = sum(Ax,0.0)/len(Ax)
        a[1][0] = sum(Ay,0.0)/len(Ay)
        a[2][0] = sum(Az,0.0)/len(Az)

        xp+=vx*deltaT
        yp+=vy*deltaT
        zp+=vz*deltaT

        Yk=[    [xp],
                [yp],
                [zp],
                [vx],
                [vy],
                [vz]]
```

```
C=np.identity(6)

Ykf=dot(C,Yk)

Xk,Pk = getState(X1,Pk1,a)

K=getKalmanGain(Pk,Xk)

Xkf=getXkf(K,Xk,Ykf)
#print (Xkf)

Pkf=getPkf(K,Pk)
#print (Pkf)

X1=Xkf
Pk1=Pkf

x2=x1+Xkf[0][0]
y2=y1+Xkf[1][0]
z2=z1+Xkf[2][0]

print("Latitude:")
print(getLatitude(x2,y2,z2)+gy-gy1)

print("Longitude:")
print(getLongitude(x2,y2,z2)+gx-gx1)

fd.write("\n\n   At Time:\t")
fd.write(str(k)+"  Millisecs")
fd.write("\nLatitude:\n")
fd.write(str(getLatitude(x2,y2,z2)+gy-gy1))
fd.write("\nLongitude\n")
fd.write(str(getLongitude(x2,y2,z2)+gx-gx1))

k=k+250
```

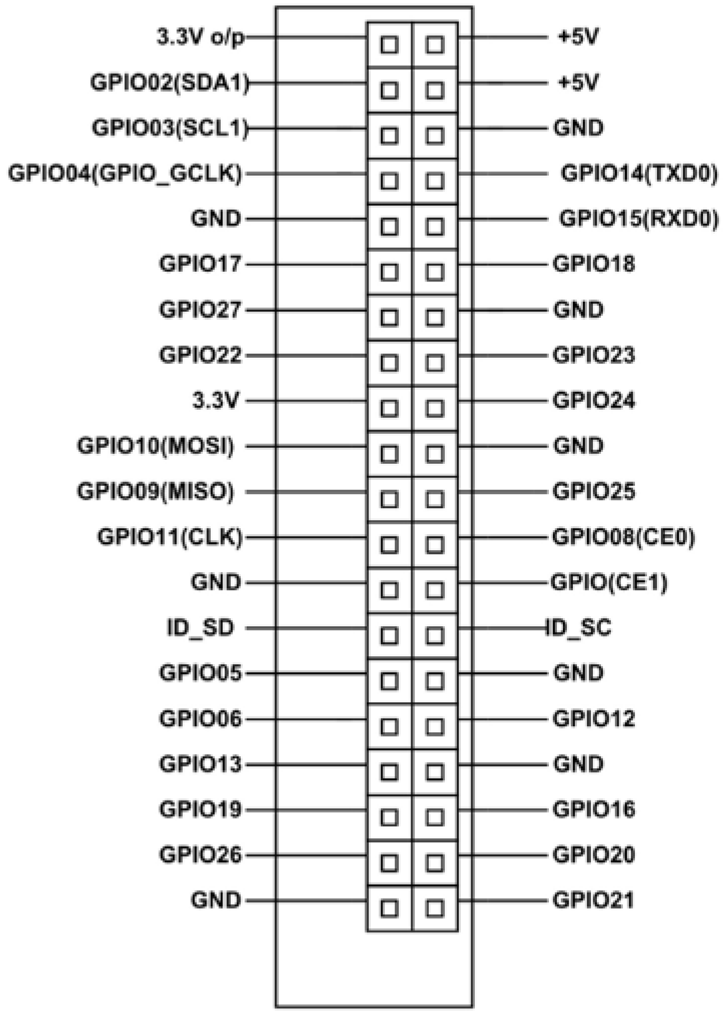
```
        sleep(0.000000001)
        """

mb.plot(t,Ax)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,Ay)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,Az)
mb.axis([0,40000,-2,2])
#mb.show()
mb.plot(t,gx)
mb.axis([0,40000,-1,1])
#mb.show()
mb.plot(t,gy)
mb.axis([0,40000,-1,1])
#mb.show()
mb.plot(t,gz)
mb.axis([0,40000,-1,1])
#mb.show()
zx = np.polyfit(t,Ax,2)
zy = np.polyfit(t,Ay,2)
zz = np.polyfit(t,Az,2)

fx = np.poly1d(zx)
fy = np.poly1d(zy)
fz = np.poly1d(zz)

zgx = np.polyfit(t,gx,2)
zgy = np.polyfit(t,gy,2)
zgz = np.polyfit(t,gz,2)

fgx = np.poly1d(zgx)
fgy = np.poly1d(zgy)
```

```
fgz = np.poly1d(zgz)

print(fx)
print(fy)
print(fz)

print(fgx)
print(fgy)
print(fgz)
g=0
while(g!=5000):
        kx1 = np.polyval(zx,g)
        ky1 = np.polyval(zy,g)
        kz1 = np.polyval(zz,g)
        kgx = np.polyval(zgx,2000)
        kgy = np.polyval(zgy,2000)
        k1 = kx1+x1
        k2 = ky1+y1
        k3 = kz1+z1
        print("\n")
        print(getLatitude(k1,k2,k3)+kgy-gy1)
        print(getLongitude(k1,k2,k3)+kgx-gx1)
        print("\n")
        fd.write("\nLatitude:\t")
        fd.write(str(getLatitude(k1,k2,k3)+kgy-gy1))
        fd.write("\nLongitude:\t")
        fd.write(str(getLongitude(k1,k2,k3)+kgx-gx1))
        fd.write("\n")
        g=g+250

print("Ax[8] : ")
print(Ax[8])
k1 = np.polyval(zx,2000)
print(k1)
print("Ay[8] : ")
print(Ay[8])
k2 = np.polyval(zy,2000)
print(k2)
print("Az[8] : ")
print(Az[8])
k3 = np.polyval(zz,2000)
print(k3)
print("gx[8] : ")
print(gx[8])
k4 = np.polyval(zgx,2000)
print(k4)
print("gy[8] : ")
print(gy[8])
k5 = np.polyval(zgy,2000)
print(k5)
print("gz[8] : ")
print(gz[8])
k6 = np.polyval(zgz,2000)
print(k6)

************************NEXT
MODULE********************

# import gmplot package
import gmplot

# GoogleMapPlotter return Map object
# Pass the center latitude and
# center longitude
gmap1 = gmplot.GoogleMapPlotter(17.393608, 78.319909, 13 )

# Pass the absolute path
gmap1.draw( "G:\DESKTOP\Final Project\\map11.html" )

**************************MODULE********************
```

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
"""
m = Basemap(projection='mill',
            llcrnrlat = 50,
            llcrnrlon = 130,
            urcrnrlat = -25,
            urcrnrlon = 70,
            resolution = 'l')

CBITlat, CBITlon =
xpt, ypt = m(CBITlon, CBITlat)
m.plot(xpt, ypt, 'c*', markersize = 15)
"""
m = Basemap(projection = 'mill', width = 100, height = 1000, lon_0 =
78.3195, lat_0 = 17.3921, resolution ='f')

m.plot(78.3195, 17.3921)

m.drawcoastlines()
m.drawcountries(linewidth = 2)
m.drawstates(color = 'b')
#m.bluemarble()

plt.title('BaseMap')
plt.show()
```

APPENDIX 2

Raspberry Pi and MPU6050 pins



Pin	Particle	Description
GPIO0		I2C data line used to identify Pi Hats (RESERVED FOR SYSTEM)
GPIO1		I2C clock line used to identify Pi Hats (RESERVED FOR SYSTEM)
GPIO2	SDA	I2C data line [2]
GPIO3	SCL	I2C clock line [2]
GPIO4	D0	Digital IO
GPIO5	D4	Digital IO
GPIO6	D5	Digital IO
GPIO7	CE1	SPI chip enable 1, digital IO
GPIO8	CE0	SPI chip enable 0, digital IO
GPIO9	MISO	SPI master-in slave-out [3]
GPIO10	MOSI	SPI master-out slave-in [3]
GPIO11	SCK	SPI clock [3]
GPIO12	D13/A4	Digital IO
GPIO13	D6	PWM-capable digital IO
GPIO14	TX	UART hardware serial transmit [1]
GPIO15	RX	UART hardware serial receive [1]
GPIO16	D14/A5	PWM-capable digital IO
GPIO17	D1	Digital IO
GPIO18	D9/A0	PWM-capable digital IO
GPIO19	D7	PWM-capable digital IO
GPIO20	D15/A6	Digital IO
GPIO21	D16/A7	Digital IO
GPIO22	D3	Digital IO
GPIO23	D10/A1	Digital IO
GPIO24	D11/A2	Digital IO
GPIO25	D12/A3	Digital IO
GPIO26	D8	Digital IO
GPIO27	D2	Digital IO

Raspberry Pi-3 Pin Configuration

PIN GROUP	PIN NAME	DESCRIPTION
POWER SOURCE	+5V, +3.3V, GND and Vin	+5V -power output +3.3V -power output GND – GROUND pin
COMMUNICATION INTERFACE	UART Interface(RXD, TXD) [(GPIO15,GPIO14)]	UART (Universal Asynchronous Receiver Transmitter) used for interfacing sensors and other devices.
	SPI Interface(MOSI, MISO, CLK,CE) x 2 [SPI0-(GPIO10 ,GPIO9, GPIO11 ,GPIO8)] [SPI1--(GPIO20 ,GPIO19, GPIO21 ,GPIO7)]	SPI (Serial Peripheral Interface) used for communicating with other boards or peripherals.
	TWI Interface(SDA, SCL) x 2 [(GPIO2, GPIO3)] [(ID_SD,ID_SC)]	TWI (Two Wire Interface) Interface can be used to connect peripherals.
INPUT OUTPUT PINS	26 I/O	Although these some pins have multiple functionsthey can be considered as I/O pins.
PWM	Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19	These 4 channels can provide PWM (Pulse Width Modulation) outputs. *Software PWM available on all pins
EXTERNAL INTERRUPTS	All I/O	In the board all I/O pins can be used as Interrupts.

Raspberry Pi 3 Technical Specifications

Microprocessor	Broadcom BCM2837 64bit Quad Core Processor
Processor Operating Voltage	3.3V
Raw Voltage input	5V, 2A power source
Maximum current through each I/O pin	16mA
Maximum total current drawn from all I/O pins	54mA
Flash Memory (Operating System)	16Gbytes SSD memory card
Internal RAM	1Gbytes DDR2
Clock Frequency	1.2GHz
GPU	Dual Core Video Core IV® Multimedia Co-Processor. Provides Open GLES 2.0, hardware-accelerated Open VG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure.
Ethernet	10/100 Ethernet
Wireless Connectivity	BCM43143 (802.11 b/g/n Wireless LAN and Bluetooth 4.1)
Operating Temperature	-40ºC to +85ºC

MPU6050 PINS

MPU6050 Pin Configuration

Pin Number	Pin Name	Description
1	Vcc	Provides power for the module, can be +3V to +5V. Typically +5V is used
2	Ground	Connected to Ground of system
3	Serial Clock (SCL)	Used for providing clock pulse for I2C Communication
4	Serial Data (SDA)	Used for transferring Data through I2C communication
5	Auxiliary Serial Data (XDA)	Can be used to interface other I2C modules with MPU6050. It is optional
6	Auxiliary Serial Clock (XCL)	Can be used to interface other I2C modules with MPU6050. It is optional
7	AD0	If more than one MPU6050 is used a single MCU, then this pin can be used to vary the address
8	Interrupt (INT)	Interrupt pin to indicate that data is available for MCU to read.

MPU6050 Features

- MEMS 3-aixs accelerometer and 3-axis gyroscope values combined
- Power Supply: 3-5V
- Communication : I2C protocol
- Built-in 16-bit ADC provides high accuracy
- Built-in DMP provides high computational power