A COMPARATIVE ANALYSIS OF LOW COST
LOCALIZATION METHODS FOR A MARIO KART
GAME APPLICATION

A Thesis
presented to
the Faculty of California Polytechnic State University,
San Luis Obispo

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science in Electrical Engineering

by
Anuranga S. Gunaratne
August 2012
COMMITTEE MEMBERSHIP

TITLE: A Comparative Analysis of Low Cost Localization for a Mario Kart Game Application

AUTHOR: Anuranga S. Gunaratne

DATE SUBMITTED: August 2012

COMMITTEE CHAIR: Dr. John Oliver, Assistant Professor

COMMITTEE MEMBER: Dr. Fred DePiero, Professor

COMMITTEE MEMBER: Dr. Bill Ahlgren, Associate Professor
Abstract

A Comparative Analysis of Low Cost Localization for an Augmented Reality Mario Kart Game Application

Anuranga S. Gunaratne

In vehicle navigation systems, it is important to have accurate estimates of vehicle positioning. The process of determining positioning is called localization. There are two methods for localization, absolute positioning and relative positioning [1]. An absolute positioning system, such as Global Positioning System (GPS), determines the localization by using external distance and heading measures. Relative localization determines the position of an object based on its previously known location and through sensors that determine speed and heading. Examples of relative positioning sensors include wheel encoders, inertial sensors and accelerometers.

The objective of this thesis is to analyze different methods to perform localization and navigation on low speed, low cost go-karts for Augmented Reality Mario Kart game application. Several different low cost localization methods (both absolute and relative positioning methods) were tested and compared in order to improve positioning information of the go-karts.

It was found that the popular solution for localization, absolution positioning using GPS, alone provided insufficient accuracy needed for this game system. Dead-Reckoning (DR), a relative positioning technique, also suffered from inaccuracies due to accumulation of measurement error when used by itself. The optimum low cost solution was achieved by using GPS and DR in conjunction with beacons to deliver the best accuracy. The improvement of this solution with four beacons over the standalone GPS solution was 3.620 meters in Root Mean Square Error (RMSE), the error criterion used in this project.
Acknowledgements

First and foremost, I would like to thank my adviser and my teacher, Dr. Oliver for agreeing to guide me through this project. You have taught me a great deal in regards to this project as well as in the classroom. This thesis project would not have been possible without your full support, resources, and project funding. Thank you for maintaining your cool and calm nature with all that I’ve put you through in regards to this project. Also, I just wanted you to know that all of your students greatly appreciate your work as a professor.

I would also like to thank Dr. DePiero for your guidance throughout this project. If it wasn’t for the critical advice that you gave, this project would not have gotten this far. It always amazed me how much you knew in all areas of engineering. I appreciate you taking time out of your busy schedule to advise me.

I’d like to thank Dr. Ahlgren for being a member of my committee. I’ve learned a lot from your classes and conference presentations.

And last but not least, I’d like to thank the most important people to me in my life, my family and friends. Mom and Dad, I don’t even know how to thank you, but I just wanted to say that I’m forever in debt for every little thing that you’ve done for me and for all the love that you have given me. To my brother, I thank you for your love and support. I want to especially thank my uncle, Anthony, for paving the way for me to come to the United States, and then to Cal Poly, and for being the guiding light of my life. None of this would’ve been possible without you. And to my friends, especially Ruby Singh and Joshua Aquino; thank you for all your support both in and outside the classroom. All of your love and support defined my character and helped me to be the person who I wanted to be.
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Geometry of Optical Triangulation [4]</td>
<td>6</td>
</tr>
<tr>
<td>Figure 2</td>
<td>System Architecture</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>GPS on a 2D plane with two satellites [9]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 4</td>
<td>effects of timing error in GPS with two satellites [9]</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>GPS triangulation with three satellites [9]</td>
<td>14</td>
</tr>
<tr>
<td>Figure 6</td>
<td>USGlobalSat EM-406A GPS receiver [11]</td>
<td>16</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Specifications of EM-406A GPS receiver [11]</td>
<td>17</td>
</tr>
<tr>
<td>Figure 8</td>
<td>A scatter plot of GPS coordinates for a given location [12]</td>
<td>17</td>
</tr>
<tr>
<td>Figure 9</td>
<td>A typical output of the EM-406A GPS Receiver</td>
<td>19</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Symmetric placement of magnets</td>
<td>21</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Hall-effect sensor mounting</td>
<td>22</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Characterization of Wheel encoding method 1</td>
<td>23</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Encoder Disc [13]</td>
<td>24</td>
</tr>
<tr>
<td>Figure 14</td>
<td>waveforms of two optical encoding sensors [13]</td>
<td>24</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Optical shaft wheel encoder [14]</td>
<td>25</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Characterization of the Optical wheel encoding method</td>
<td>27</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Honeywell HMC6352 Digital Compass [15]</td>
<td>28</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Melexis MLX60906 Gyroscope [16]</td>
<td>29</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Gyroscope characterization and testing</td>
<td>30</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Standard Kalman Filter Equations and Process [8]</td>
<td>31</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Kalman Filter Performance</td>
<td>32</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Interface for displaying heading information</td>
<td>33</td>
</tr>
<tr>
<td>Figure 23</td>
<td>GUI with buttons for beacons</td>
<td>35</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Arduino UNO Microcontroller [18]</td>
<td>36</td>
</tr>
<tr>
<td>Figure 25</td>
<td>Data flow diagram</td>
<td>37</td>
</tr>
<tr>
<td>Figure 26</td>
<td>Results of the GPS only method that didn’t provide sufficient accuracy</td>
<td>39</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Results of Hall-effect encoding and Compass heading method that had inaccuracies in distance and heading measurements</td>
<td>41</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Results with rear mounted compass</td>
<td>42</td>
</tr>
<tr>
<td>Figure 29</td>
<td>Results of Optical wheel encoding and Compass heading method that provided accurate distance measurements but wrong headings</td>
<td>43</td>
</tr>
<tr>
<td>Figure 30</td>
<td>Results of optical wheel encoding and gyroscope heading method that shows inaccuracies in heading measurements</td>
<td>44</td>
</tr>
<tr>
<td>Figure 31</td>
<td>Results of Optical wheel encoding and Kalman Filter heading method that shows an inaccurate heading output from the filter</td>
<td>46</td>
</tr>
<tr>
<td>Figure 32</td>
<td>Results of the GPS heading and Dead-Reckoning implementation that provided the desirable accuracies</td>
<td>48</td>
</tr>
<tr>
<td>Figure 33</td>
<td>Results with two equally spaced beacons</td>
<td>50</td>
</tr>
<tr>
<td>Figure 34</td>
<td>Results with three equally spaced beacons</td>
<td>50</td>
</tr>
</tbody>
</table>
Figure 35: Results from second configuration of the beacons that provided high accuracy with low number of beacons ................................................................. 51
Figure 36: The Ideal Path .................................................................................................................... 52
Figure 37: Equation for RMS Error .................................................................................................. 53
Figure 38: RMSE values of different implementations .................................................................... 54
Figure 39: RMSE values of beacons placed at equal distances ..................................................... 56
Figure 40: LTC3824 controller for Linear Technology [21] ............................................................ 59
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>GPS Accuracy Measures [12]</td>
<td>18</td>
</tr>
<tr>
<td>Table 2</td>
<td>Characterization of Wheel encoding method 1</td>
<td>22</td>
</tr>
<tr>
<td>Table 3</td>
<td>Outputs of two optical wheel encoding sensors [13]</td>
<td>24</td>
</tr>
<tr>
<td>Table 4</td>
<td>Characterization of optical wheel encoding method</td>
<td>25</td>
</tr>
<tr>
<td>Table 5</td>
<td>Averages values from the characterization of optical wheel encoding</td>
<td>26</td>
</tr>
<tr>
<td>Table 6</td>
<td>RMSE values of different implementations</td>
<td>54</td>
</tr>
<tr>
<td>Table 7</td>
<td>RMSE values of beacons placed at equal distances</td>
<td>55</td>
</tr>
<tr>
<td>Table 8</td>
<td>RMSE values from the second configuration of beacons</td>
<td>57</td>
</tr>
</tbody>
</table>
Chapter 1 – INTRODUCTION

The target application of this project is an Augmented Reality (AR) Mario Kart Game Application. Mario Kart is a highly popular video game created by Nintendo where players are able to select one of their favorite Mario characters and compete in racing tournaments. In AR Mario Kart, players will drive real go-karts and wear Virtual Reality (VR) goggles that overlay game graphics on reality, and interactions with game mechanics are controlled via a central computer. Each of the Mario karts will be equipped with a laptop for the primary purposes of producing game graphics, and communicating with the central computer via a wireless connection. The karts are also equipped with sensors and micro-controllers that aid the localization and controlling of the kart according to the game mechanics.

For this game system to function very well, it is essential to know the location of the karts as accurately as possible at any given time. For this reason, this thesis has been completely dedicated to investigate localization of the kart. The objective of this thesis is to analyze different methods to perform localization and navigation on low speed, low cost go-karts for this game application. It was also important that low cost localization systems are explored.

This document begins with background information of different localization techniques and their applications. Then it discusses the explored localization methods and various sensors used. In the testing and results chapter, this document will describe the performance of each method and a comparison between them, including a final recommended solution for localization of these karts. The document is then concluded with final remarks and possible improvements that could be done for this project in the
conclusion and future work chapter. The appendices contain various supporting documents including code that was used in this project written in C and Python programming languages.
Chapter 2 – BACKGROUND

There are many existing technologies and algorithms used for localization. This chapter briefly discusses these technologies and algorithm and presents the most suitable of these for this project.

Section 2.1 – Hardware Technologies used for Localization

Localization is the process of locating an object of interest within a given a priori infrastructure such as a map. It is essential for a vehicle navigation system to have an accurate estimate of the position of the vehicle on this given map. Likewise, to the Mario Kart game system, it is very important to know the location of each kart on the race track.

A complete localization system generally has two main components. The first component is the sensors used to measure and characterize environment the system is in. The second component is the algorithms used to estimate its location based on the calculations performed on sensor measurements.

The first component comprises of mechanical and/or electrical sensors placed on and off the vehicle. These sensors are categorized by methods of localization that they perform. There are two general methods used for localization: relative positioning and absolute positioning [1]. Absolute positioning is to determine the location of an object using external distance and heading measures. A very popular and a heavily used absolute positioning system for localization is Global Positioning System (GPS).

The GPS system is a satellite based navigation system made up of a network of 24 satellites placed into orbit by the U.S. Department of Defense. GPS satellites circle the earth in a very precise orbit and transmit signal information to earth. Passive devices on
earth, such as GPS receivers, take this information and use triangulation to calculate the receiver’s exact location. The GPS receiver compares the time a signal was transmitted by a satellite with the time it was received to calculate the distance to the satellite. To have a very accurate time reference, each GPS satellite is equipped with precise atomic clock. To determine the time-of-flight of a signal, GPS satellite transmits data that includes its location and the time at signal transmission. The GPS receiver performs triangulation by taking distance measurements (by calculating signal time-of-flight) from a few more satellites.

A GPS receiver must be locked on to at least three satellites to calculate its 2 dimensional position (latitude and longitude) and track movement. For a 3 dimensional position (latitude, longitude, and altitude), the receiver needs to track four or more satellites. By constantly recalculating its position, the GPS receiver can determine the vehicle speed and the direction of movement. Today’s multi-channel GPS receivers have the ability to track several satellites for increased accuracy.

GPS’s popularity suggests that using GPS technology to be the easiest for localization. As long as a GPS signal is attainable, this technology can help locate an object down to an accuracy of few meters. GPS signals work on the line of sight principle. Therefore a clear line of sight should be maintained between the satellites and the receiver for the results to be accurate. Due to this reason, GPS systems experience degraded performance in areas with tall obstructions such as a city with tall buildings. Yet, due to emerging advances in GPS technology and its significant reduction in cost over the years, it is the most commonly used method of localization in today’s automotive industry. Advances such as Wide Area Augmentation System (WAAS) which
transmits GPS correction messages to account for satellite orbit and clock drift pulse delays caused by atmosphere and ionosphere have helped this technology to achieve an accuracy of less than 3 meters [2]. Due to the nature of Mario Kart project and its similarities with an automotive environment, it was sensible to use this low cost localization method.

Computer vision systems are also used for localization. Here, the a priori infrastructure is created by building a database of images taken along a path of interest. These images are also tagged with GPS coordinates of the location where they were captured [3]. Using this database a map is created. Localization is performed by comparing an image captured using a camera mounted on the vehicle, against the database of images. In case of a match, the current location of the vehicle would be the GPS coordinates of matching image from the database.

Two other methods used for localization and navigation are Sonar and Radar technologies. Sonar uses propagation of sound waves and their echo to measure distances to surrounding objects. By continuously sending and receiving acoustic signals, this technology can map the surroundings and estimate the position of the vehicle relative to the surrounding objects. Sonar works best in underwater applications where sound wave propagation is better. Radar works in a similar way to Sonar but uses radio waves and their reflections instead of acoustic signals. Radar works reliable on the ground, and in the sky. Therefore, Aircrafts and some automobiles equipped with adaptive cruise control uses this technology to measure distances from objects and obstacles in front of the vehicle. A reliable implementation of this technology is very costly and is beyond the scope of this project.
Laser Rangefinder based systems are also used in various localization applications. This technology determines distances to objects by emitting a laser beam and taking measurements on the reflected beam. There are three general ways of taking distance measurements in this technology. They are time-of-flight, Phase shift, and optical triangulation. Time-of-flight uses the time difference between the emitted and reflected beams, and the speed of light through the medium to calculate distances. The Phase shift method used the phase shift between the emitted and reflected beams to measure distances. In this case, the distances that can be measured are limited by the wavelength of the signal. And lastly, optical triangulation is used where the reflected beam is focused on to a sensor using optics. The location of the light spot on the sensor corresponds to the distance to the object [4]. Following figure shows a typical layout of equipment for optical triangulation.

![Geometry of Optical Triangulation](image)

**Figure 1: Geometry of Optical Triangulation [4]**

Laser Rangefinders are very accurate. But like Sonar and Radar systems, Laser Rangefinders too are costly and are beyond the scope of this project.

The second general method used for localization is relative positioning. Relative positioning, also known as dead-reckoning, evaluates the current position of a vehicle
using its displacement and yaw angles measured by encoders attached to the wheels or by inertial sensors. Dead reckoning estimates a relative position from the initial starting point information [1]. To measure displacement, optical shaft encoders are used that measure the rotation of the wheel or a motor. Yaw angles, which tell the direction of the heading of the vehicle, could also be measured using two or more encoders or different wheels of the vehicle. Inertial sensors such as gyroscopes are also very in measuring the direction of heading as they measure the rate of rotation of the vehicle. Dead-reckoning systems are completely self-contained and they are always capable of providing the vehicle with an estimate of its position [5]. Therefore, dead-reckoning has advantages in being simple, low cost, and has an easier time in estimating the position in real time compared to absolute positioning methods [1]. The main disadvantage is that the position and orientation errors grow without bounds [5]. [1] And [5] discusses implementing navigation systems for mobile robots using dead-reckoning systems. To stop the error from growing out of bounds, dead-reckoning systems are usually coupled with an absolute positioning system. This is the case in an automotive environment where a dead-reckoning system is coupled with GPS technology. Low cost inertial sensors and systems are motivated by the needs of the automotive industry and are increasingly being made commercially available [5]. Due to the major similarities between this Mario Kart project and a typical automotive application, and also due to the low cost, a dead-reckoning implementation was performed for this project.

**Section 2.2 – Algorithms used for Localization**

The previous section discussed hardware background for localization. There are also algorithms that aid the hardware implementations for localization. These algorithms
are typically implemented using software or low level firmware in micro-controllers for embedded applications. Two of the most popular algorithms are Particle Filter, and Kalman Filter.

Particle filter is based on sequential Monte Carlo approach (as an implementation of Bayes’ theorem), where the probability density in represented by a set of weighted samples (called particles) [6]. It tracks a variable or an object of interest as it evolves over time. For an implementation of a Particle filter, first a set of particles are generated to represent the starting state. Then, control inputs with noise are applied to all particles using a state transition model. Any type of function can be used as the state transition model as long as it generates propagated state vector base on the inputs of a previous state and a control input vector. Particles that are situated in locations in the state space where the expected measurements and the real measurements are similar receive a higher weight [7]. Which particles are to be resampled is determined by the weight of the particles. In this way, the Particle filter implements a survival of the fittest technique. In the recent years, particle filter has been successfully applied in computer vision community for visual tracking [6]. Since particle filters are to be applied with computer vision systems, this algorithm wasn’t implemented of the Mario Kart project as the environmental factors and the cost factors are not aligned with the project goals.

Kalman filter represents a set of mathematical equations that provides an efficient computational (recursive) means to estimate the state of a process, in a way that minimizes the mean of the squared error [8]. This filter operates in the framework of Gaussian probability density functions. The state model is built as a linear function of the last state and Gaussian noise, and the sensor (measurement) model is built as a linear
function of state and Gaussian noise of the sensor signal. The filter algorithm estimates the state and attempts to correct it using the measurement updates recursively. Since the time R.E. Kalman published his paper describing a recursive solution to the discrete-data linear filtering problem, Kalman filter has been the subject of extensive research and application, particularly in the area of autonomous or assisted navigation [8]. Today, Kalman filter is heavily used in automotive industry for vehicle navigation systems. With the advances of digital computing, this recursive algorithm could easily be implemented at a low cost micro-controller level platform. Due to its wide use in the automotive industry, and its low cost implementation, this algorithm was implemented for this thesis study within the Mario Kart game system.
Chapter 3 – Problem Definition and Implementation of Solution(s)

A localization system is needed by each kart of the Augmented Reality (AR) Mario Kart game to let the game server know their locations on the race track. The effectiveness of the game depends heavily on the performance of the localization systems. Therefore, the objective of this thesis is to analyze different methods to perform localization and navigation on the low speed and low cost go-karts that are used for the AR Mario Kart game. There are no set parameters that define the limitations or requirements of this project. However, it is important that the solution be a low cost implementation that can achieve a greater accuracy.

This chapter discusses the different methods that were implemented to perform localization of the go-karts. Each of the following sections present the different methods and the equipment used to implement each method. Some preliminary test results are also presented in these sections, such as results that predict Optical Shaft Wheel Encoding to have a higher accuracy than the Hall-effect sensor wheel encoding method. However, the overall results from each implementation are presented in Chapter 4.

Section 3.1 - Overview of the implementation and System Architecture

The investigation for a suitable localization system begins with the popular absolute positioning system, GPS, as it is the easiest to implement. Thereafter, relative positioning systems are implemented by themselves and also in conjunction with absolute positioning systems in order to create a system with high accuracy. The following list presents the overview of the implementation.
• GPS Only (Section 3.2)
• Dead-Reckoning (DR) Only (Section 3.3)
  ▫ Wheel Encoding Methods (Section 3.3.1)
    • Magnets + Hall-effect Sensor
    • Quadrature Shaft Encoding (Optical)
  ▫ Heading measuring methods (Section 3.3.2)
    • Gyroscope Only
    • Compass Only
    • Gyroscope + Compass (Kalman Filter implementation)
• GPS Heading + Dead-Reckoning (Section 3.4)
• Beacons + GPS Heading + Dead-Reckoning (Section 3.5)

Note: - The Dead-Reckoning (DR) Only implementation is a combination of one wheel encoding method and one heading measuring methods.

Figure 2: System Architecture
Section 3.2 – GPS Only Implementation

Introduction

As it was discussed in Chapter 2, Global Positioning System (GPS) is a satellite base navigation system. These satellites circle the twice a day in a very precise orbit and transmit the signal to earth. GPS receivers take this signal and use triangulation to calculate its position. The Mario Kart project requires the maxim accuracy that a certain low cost implementation can provide. Therefore, it is important to discuss the accuracy and the sources of error of the implementation.

GPS receivers compare the time that the signal was transmitted by a satellite to the time that it was received to calculate the distance to the satellite. Measuring this time difference has to be done very accurately. GPS satellites are equipped with very precise atomic clocks. Due to their high price, atomic clocks are not used in GPS receivers. GPS receivers use conventional quartz clocks which are comparative inaccurate.

Imagine the earth to be a two dimensional disk. If one satellite is communicating with the GPS receiver and the signal measurement shows a 4 second time difference, then the location of the GPS receiver falls on the circle of radius of 4 seconds. The circle on the left of the figure below shows this. If we perform the same signal measurement with the second satellite and measured a time difference of 5 seconds, then the location of the receiver falls on the circle of radius of 5 seconds from the second satellite. The circle on the right of the figure below shows this. However the collective result of these measurements suggests that the GPS receiver should be on one of the two intersections, points A or B of the figure below.
Figure 3: GPS on a 2D plane with two satellites [9]

With the assumption that the GPS receiver cannot exist in deep space, we can ignore point B and assume that the location of the receiver is at point A. However, a minor error in measuring the time difference could lead to a major error in determining the location of the receiver.

Imagine the receiver’s clock is 0.5 seconds early compared to the clock in the satellite leading to the position estimate of B as shown in the figure below.

Figure 4: effects of timing error in GPS with two satellites [9]
A clock error of 1/100 of a second would in GPS navigation lead to an error in the position of about 3000 km. To achieve an accuracy of 10m of the position, the signal must be precise to 0.00000003 seconds. For this reason a third satellite is used to determine the location of the receiver, hence the term triangulation. If the signal of the third satellite measures a 6 second time difference, a third circle of radius 6 seconds is added to the picture. By including a third satellite in the tracking process, in the event of a time measurement error, the error stands out immediately as by points B on the figure below.

![Figure 5: GPS triangulation with three satellites](image)

The time of the GPS receiver’s clock is shifted until the three intersection points B merge to A, hence synchronizing the receiver’s clock with the clocks of the satellites.

As discussed above, and two dimensional position determination, also known as a 2D-fix, is achieved by tracking three satellites. The results of a 2D-fix are latitude and
longitude. By tracking a fourth satellite, a three dimensional position determination or a 3D-fix could be achieved. In the case of a 3D-fix, the results would be latitude, longitude as well as altitude. By constantly recalculating its position, GPS receiver can additionally determine the speed and direction of its movement. Three of the resulting parameters, latitude, longitude, and direction of movement (also known as heading), are used for localization in this project.

Following are some of the sources of error in GPS technology as stated in [10],

- **Ionosphere and troposphere delays** - The slowing of satellite signal as it passes through the atmosphere. A built-in model partially corrects for this type of error.
- **Signal multipath** - This occurs when the GPS signal is reflected off objects such as tall buildings or large rock surfaces.
- **Receiver clock errors** - A receiver’s clock is not accurate as the atomic clocks onboard the GPS satellites.
- **Number of satellites visible** - The more satellites a GPS receiver can see, the better the accuracy. Buildings and terrain can block the signal.
- **Satellite geometry/shading** - Refers to the relative position of the satellites at any given time. Ideal satellite geometry is when the satellites are located at wide angles relative to each other. Poor geometry is when they’re located in a line or in a tight group.
- **Intentional degradation of satellite signals** - Selective Availability (SA) is an intentional degradation of the signal imposed by the U.S. Department of Defense and was removed in May 2000.
**GPS Hardware and software**

The GPS receiver EM-406A from USGlobalSat was used for this project. The following figure shows this GPS receiver,

![GPS Receiver](image)

**Figure 6: USGlobalSat EM-406A GPS receiver [11]**

Throughout this project, special attention is paid to the cost of the equipment used as one of the main project goals is to create a low cost localization solution for this Mario Kart game application. This GPS receiver is equipped with a built-in patch antenna, a super capacitor to power data storage, and internal voltage regulators to power this receiver. This eliminates the cost needed to get an external antenna (passive or active), and external voltage supplies to power the RAM (Random Access Memory for data storage) as well as voltage conditioning requirements when powering the receiver.

Figure 5 shows the specifications of this receiver. The important specification here is the position accuracy. This module has 10 meters 2D-RMS accuracy and 5 meters 2D-RMS accuracy when WAAS (Wide Area Augmentation System) is enabled. When GPS positions are logged over time, the positions are scattered over an area due to measurement error. This dispersion of points is called a scatter plot (as shown on Figure 6), which GPS manufacturers use to characterize their equipment’s accuracy [12].
**Specifications:**

**General**
- Chipset: SiRF Star III
- Frequency: L1, 1575.42 MHz
- C/A code: 1.023 MHz chip rate
- Channels: 20 channel all-in-view tracking
- Sensitivity: -159 dBm

**Accuracy**
- Position: 10 meters, 2D RMS
  - 5 meters, 2D RMS, WAAS enabled
- Velocity: 0.1 m/s
- Time: 1ns synchronized to GPS time

**Datum**
- Default: WGS-84

**Acquisition Time**
- Reacquisition: 0.1 sec., average
- Hot start: 1 sec., average
- Warm start: 38 sec., average
- Cold start: 42 sec., average

---

**Figure 7: Specifications of EM-406A GPS receiver [11]**

**Figure 8: A scatter plot of GPS coordinates for a given location [12]**

DRMS is a single number that expresses 2D accuracy. In order to compute the DRMS of horizontal position errors, the standard error (denoted by σ) from the known
position in the directions of the coordinate axis are required. DRM is the square root of the average of the square errors which is defined as,

$$DRMS = \sqrt{\sigma_x^2 + \sigma_y^2}$$

The following table describes some of the most common position accuracy measures used by GPS manufacturers. The probabilities for each case shown on Table 1 are referring to the area of an ellipse or a circle in which the stated percentage (under the probability column) of samples would fall on. For example, CEP refers to the radius of a circle in which 50% of the data points will fall on from a given data set.

**Table 1: GPS Accuracy Measures [12]**

<table>
<thead>
<tr>
<th>Accuracy Measures</th>
<th>Formula</th>
<th>Probability</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRMS</td>
<td>$\sqrt{\sigma_x^2 + \sigma_y^2}$</td>
<td>65%</td>
<td>The square root of the average of the squared horizontal position errors.</td>
</tr>
<tr>
<td>2DRMS</td>
<td>$2\sqrt{\sigma_x^2 + \sigma_y^2}$</td>
<td>95%</td>
<td>Twice the DRMS of the horizontal position errors.</td>
</tr>
<tr>
<td>CEP</td>
<td>$0.62\sigma_y + 0.56\sigma_x$ (Accurate when $\sigma_y / \sigma_x &gt; 0.3$)</td>
<td>50%</td>
<td>The radius of circle centered at the true position, containing the position estimate with probability of 50%</td>
</tr>
<tr>
<td>R95</td>
<td>$R(0.62\sigma_y + 0.56\sigma_x)$ ($R=2.08$, when $\sigma_y / \sigma_x = 1$)</td>
<td>95%</td>
<td>The radius of circle centered at the true position, containing the position estimate with probability of 95%.</td>
</tr>
</tbody>
</table>

The EM-406A module uses UART (Universal Asynchronous Receiver/Transmitter) as its communication protocol. It outputs NMEA 0183 standard statements through its serial ports which is read by the Arduino microcontroller. The following are the output statements of this module,

- **$GPGGA** – Global Positioning Systems Fix Data
- **$GPGSA** – GPS DOP and active satellites
- **$GPGSV** – GPS satellites in view
$GPRMC – Recommended minimum specific GPS/Transit data

Of these statements, only parameters embedded in the $GPGGA and $GPRMC are used for this project. A typical output of this module is shown in the Figure 8 below. The highlighted (in red) fields are the values used for this project.

\[
\begin{align*}
\text{\$GPGGA,050230.000,3518.0442,N,12039.8055,W,1,08,1.0,101.3,M,-30.6,M,0000*64} \\
\text{\$GPGSA,A,3,27,04,12,02,15,09,17,28,,1.8,1.0,1.5*3F} \\
\text{\$GPRMC,050230.000,A,3518.0442,N,12039.8055,W,0.63,164.90,290412,,A*77} \\
\text{\$GPGGA,050231.000,3518.0439,N,12039.8053,W,1,08,1.0,101.0,M,-30.6,M,0000*6C} \\
\text{\$GPGSA,A,3,27,04,12,02,15,09,17,28,,1.8,1.0,1.5*3F} \\
\text{\$GPRMC,050231.000,A,3518.0439,N,12039.8053,W,1.39,166.54,290412,,A*78} \\
\text{\$GPGGA,050232.000,3518.0435,N,12039.8050,W,1,08,1.0,100.7,M,-30.6,M,0000*66} \\
\text{\$GPGSA,A,3,27,04,12,02,15,09,17,28,,1.8,1.0,1.5*3F} \\
\text{\$GPGSV,3,1,12,27,65,258,41,04,60,127,39,09,57,290,40,02,41,179,37*70} \\
\text{\$GPGSV,3,2,12,17,36,046,30,12,30,307,33,28,25,096,22,15,11,239,32*72} \\
\text{\$GPGSV,3,3,12,10,02,134,,26,02,199,,20,01,051,,48,46,200,37*77}
\end{align*}
\]

**Figure 9: A typical output of the EM-406A GPS Receiver**

Of the parameters presented in the $GPGGA statement, *Latitude* and *Longitude* are used for localization in this project. All the parameters of the $GPGGA statement are shown below,

1  = UTC of Position  
2  = Latitude  
3  = N or S  
4  = Longitude  
5  = E or W  
6  = GPS quality indicator (0=invalid; 1=GPS fix; 2=Diff. GPS fix)  
7  = Number of satellites in use [not those in view]  
8  = Horizontal dilution of position  
9  = Antenna altitude above/below mean sea level (geoid)  
10 = Meters (Antenna height unit)  
11 = Geoidal separation (Diff. between WGS-84 earth ellipsoid and mean sea level. geoid is below WGS-84 ellipsoid)  
12 = Meters (Units of geoidal separation)  
13 = Age in seconds since last update from diff. reference station  
14 = Diff. reference station ID#  
15 = Checksum
Of the parameters presented in the $GPRMC$ statement, *Speed over ground in knots* and *Track made good in degrees true* are used in this project. All the parameters of the $GPRMC$ statement are shown below,

1    = UTC of position fix
2    = Data status (V=navigation receiver warning)
3    = Latitude of fix
4    = N or S
5    = Longitude of fix
6    = E or W
7    = Speed over ground in knots
8    = Track made good in degrees True
9    = UT date
10   = Magnetic variation degrees (Easterly var. subtracts from true course)
11   = E or W
12   = Checksum

**Section 3.3 Dead-Reckoning (DR) Only**

**Introduction**

As discussed in the background section of this thesis, Dead-Reckoning (DR) is a relative positioning implementation as it calculates the next position based on the previous position and sensor data. The wheel encoding methods used for this project are Hall-effect sensor plus magnets, and optical shaft wheel encoding. These methods measure the displacement of the Kart as well as its speed. To measure direction of travel (Heading), a gyroscope and a digital compass was used.
Section 3.3.1 – Wheel Encoding methods

Wheel encoding using Hall-effect sensing

This type of wheel encoding is used in harsh environments where there are a lot of moving parts in places where wheel encoding is needed to be implemented. Also the placement of the Hall-effect sensor has the greatest flexibility of the two wheel encoding methods implemented for this project. The cost of this implementation is minute compared to many other encoding methods. The Hall-effect sensor and each of the magnets costs less than a dollar at the time of this writing.

In this implementation, six magnets are placed around the rim of the wheel at equal distances. A Melexis US1881 Hall-effect sensor was used to detect these magnets and interrupt the micro-processor. These interrupts will cause a counter to increment which is then used to calculate the displacement of the Kart. The implementation of this method is shown in the following Figures 9 and 10,

![Image of symmetric magnetic wheel placement](image1.jpg)

**Figure 10: Symmetric placement of magnets**

Six magnets are placed at equal distances from each other around the rim of the tire as show on the figure above. The following figure shows the wheel with the placement of magnets installed in the car. It also shows a metal arm (circled in red) on to which the Hall-effect sensor is mounted.
Two types of tests were performed on the wheel encoding method. The first one was to push the kart a known distance and compare it with the distance that the system has calculated as the displacement. The sources errors eliminated here are wheel slippage, and the sagging of the wheel when weight is introduced to the wheel. The second test was to lift the drive wheels of the car above the ground level to have them freely turn, and measure the displacement calculated by the system. These tests were performed in a lab on a smooth surface to minimize the error caused by the terrain. Each revolution of the wheel is a displacement of 795.45mm. The theoretical displacement was compared with the data from the system. Following are the results,

<table>
<thead>
<tr>
<th>Revolutions</th>
<th>Encoder Distance (m)</th>
<th>Real Displacement (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.67</td>
<td>0.79545</td>
</tr>
<tr>
<td>2</td>
<td>1.33</td>
<td>1.5909</td>
</tr>
<tr>
<td>4</td>
<td>2.66</td>
<td>3.1818</td>
</tr>
<tr>
<td>8</td>
<td>5.33</td>
<td>6.3636</td>
</tr>
<tr>
<td>16</td>
<td>10.66</td>
<td>12.7272</td>
</tr>
</tbody>
</table>
As seen from the data in Table 2, the system measures approximately 125cm less with every revolution when compared to the actual displacement. The resolution of the system could be a contribution to this discrepancy. The ideal case would be to use smaller, closely spaced magnets as it would provide better resolution. Another error would be magnets that are not placed precisely at equal distances from each other. For the placement of these magnets, the rubber tabs that were left by the tire manufacturer were used, and the distances between each magnet were then measured. The Hall-effect sensor’s placement is also vital to the performance of this implementation.

**Wheel encoding using Optical Shaft Wheel Encoding**

A Quadrature type Optical Shaft encoder consists of a Gray coded disk and two optical sensors. The disk is coded in black and white in concentric circles such that the difference between two neighboring sections would be a difference of one bit. An image of a Gray coded circle is shown on figure 9,
The bits that the two sensors read and their waveforms are shown below,

**Table 3: Outputs of two optical wheel encoding sensors [13]**

<table>
<thead>
<tr>
<th>Phase</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 13: Encoder Disc [13]**

Outputs of the two optical sensors are out of phase by 90 degrees, and hence the name Quadrature. The advantages of optical shaft wheel encoding over the previously discussed wheel encoding implementation are that its high resolution as well as being
able to detect the direction of rotation (clockwise or counter-clockwise) by noting the sensor that leads in the waveform.

A 200 pulse per revolution shaft encoder was used for this project. The following figure shows the shaft encoder.

Figure 15: Optical shaft wheel encoder [14]

Despite the difficulties in mounting this encoder such that it’s in line with the drive shaft of the wheel, the encoder provided very accurate readings. The outputs from the two optical sensors inside this encoder were used as interrupts for the processor of the Arduino micro-controller. These interrupts would cause an increment or a decrement to the software counter implemented in the micro-controller depending on the direction of rotation of the wheel.

This wheel encoding implementation was put through the same tests as the previous method of wheel encoding. The following are the data from the tests for this implementation,

<table>
<thead>
<tr>
<th>Revolutions</th>
<th>Actual Displacement (m)</th>
<th>Encoder Ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.79545</td>
<td>199</td>
</tr>
<tr>
<td>1</td>
<td>0.79545</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>0.79545</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td>0.79545</td>
<td>201</td>
</tr>
<tr>
<td>1</td>
<td>0.79545</td>
<td>201</td>
</tr>
</tbody>
</table>
Noting that each revolution of the wheel was to trigger 200 ticks, this wheel encoder performed at very high accuracy of up to +/- 2 ticks. The average values of these results (in terms of displacement in meters) is shown below.

<table>
<thead>
<tr>
<th>Revolutions</th>
<th>Nominal Distance (m)</th>
<th>Encoder Distance (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.79545</td>
<td>0.79624545</td>
</tr>
<tr>
<td>2</td>
<td>1.5909</td>
<td>1.5861273</td>
</tr>
<tr>
<td>4</td>
<td>3.1818</td>
<td>3.1770273</td>
</tr>
<tr>
<td>8</td>
<td>6.3636</td>
<td>6.3651909</td>
</tr>
<tr>
<td>16</td>
<td>12.7272</td>
<td>12.7272</td>
</tr>
</tbody>
</table>
The resolution of this implementation could be increased even higher by using a wheel encoder that has higher number of pulses per revolution. However, due to the goal of achieving a low cost solution, the 200 pulse per revolution was chosen.

Section 3.3.2 - Methods of measuring Heading

Measuring heading using the Digital Compass

Aside from the GPS receiver, the only other absolute positioning sensor that was used in this project was a HMC6352 digital compass from Honeywell. The compass is shown on the following image,
This module measures the earth magnetic field and provides a bearing from the magnetic north in degrees through its I2C (Inter-Integrated Circuit) communication protocol. This compass has a resolution of up to 1/10\textsuperscript{th} of a degree, and is a very simple drop-in solution if a quick heading measurement is needed. This module seemed to detect the earth’s magnetic north accurately as long as it was away from any magnetic interference. However, at the presence of a certain magnetic interference, even if it is minor, the compass provided erratic data. Since many parts of the Kart, including the frame, was metal, mounting this sensor on the Kart was an issue as anything metal would interfere with the readings of this sensor. Also, since the kart was powered by an electric motor, if the sensor was mounted in close proximity to the motor, it would not read the earth’s magnetic field.

**Measuring heading using the Gyroscope**

A gyroscope measures the angular velocity about an axis of interest. For this implementation the Melexis MLX90609 single axis gyroscope capable of measuring up to 300 degrees/second of angular velocity was used. An image of this sensor is shown below,
As this is a relative positioning sensor, the initial heading has to be known to measure heading with this sensor. The initial heading is usually a known parameter since a race in Mario Kart starts with the Karts facing a certain direction at the start line. This initial heading could be measured from a compass and hard coded into the system as the initial bearing of the system.

This sensor was read at a rate of approximately 50Hz through its SPI (Serial Peripheral Interface bus) and the values were added to obtain an angular displacement from the previous kart heading. One major source of error for this sensor is caused by the effects physical vibrations to this sensor. Since the Karts that were used for this project didn’t have a suspension, the vibrations caused by driving on the terrain are directly passed on to the localization systems without much dampening. This causes the sensor to detect these vibrations as movement. Software filters were used to ignore this type of reading. These filters are simple conditions to ignore low level readings and were varied depending on the terrain. The other major source of error is that since this is a relative positioning sensor, the error is accumulated over time as it measures angular displacement relative to the prior headings. This error could grow out of bounds if the measurement is not initialized back to a known heading.
Due to the absence of a solid low voltage (5V) power supply, this sensor had to be calibrated through software fixes to get it to its linear response range. Simple tests were done such as the one shown in the following figure where the sensor was rotated about an axis at varying angles to match the sensor measurement with the actual angular displacement.

![Image of sensor characterization and testing](image)

**Figure 19: Gyroscope characterization and testing**

**Kalman Filter for heading measurement**

The Kalman filter estimates a process by using a form of feedback control: the filter estimates the process state at some time and then obtains feedback in the form of (noisy) measurements. As such, the equations for the Kalman filter fall into two groups: time update equations and measurement update equations. The time update equations are responsible for projecting forward (in time) the current state and error covariance estimates to obtain the a priori estimates for the next time step. The measurement update equations are responsible for the feedback – i.e. for incorporating a new measurement into the a priori estimate to obtain an improved a posteriori estimate [8].
The state that the Kalman filter is designed to estimate in this project is the heading of the vehicle. To achieve this, a two state Kalman filter was used to combine the outputs of the digital compass and the gyroscope to obtain a more accurate estimation for heading. The primary state for this filter was angle and measurement update for this state was from the digital compass. The secondary state is angular velocity and the measurement update for this state was from the gyroscope.

Therefore the state vector for this filter is,

\[ X = [\text{angle}, \text{gyro\_bias}] \]

Due to the magnetic interference experienced by the digital compass, the process covariance noise values were adjusted such that the values from the gyroscope were trusted more than the values from the compass. The noise value for compass was 0.003 and the noise value for gyroscope was 0.001. These values were stored in the process covariance noise matrix \( Q \) in the following equations. The standard Kalman filter equations are shown below,

![Figure 20: Standard Kalman Filter Equations and Process [8]](image-url)
Due to the absence of a control signal, the term $u_{k-1}$ is ignored. According to implementation in [17], ‘A’ matrix was taken to be $[0\ -1\ 0\ 0]$, and the ‘H’ matrix was $[1\ 0\ 0\ 1]$. The covariance matrix ‘P’ was initialized to be $[1\ 0\ 0\ 1]$, and R, the measurement covariance noise, was observed to be 0.3. Due to the lack of floating point operation capability in the processor of an Arduino microcontroller, code optimization was done according to [17], to perform faster computations. The following shows a plot of filter performance,

![Kalman Filter Performance](image)

**Figure 21: Kalman Filter Performance**

The Kalman filter output (in red) factors in both the output from the compass (in blue) as well as the output from the gyroscope (in green), and determines the best estimate for the heading of the vehicle. From the above graphs, it could be seen that the filter rejects the jitter and sudden changes in the outputs of the two sensors and outputs a smooth estimate for the heading.

For visual observation of the readings and sensor orientation of the digital compass, the gyroscope, and the Kalman filter output, the following user interface (UI) was developed.
using python scripting language and Nokia’s QT library. The following shows a screen shot of this UI,

![Screen shot of UI](image)

**Figure 22: Interface for displaying heading information**

**Section 3.4 - GPS Heading with DR**

Magnetic interference created by the kart was a major concern in using the digital compass as a sensor to measure the heading of the kart. Both the chassis and the electric motor create magnetic fields that could result unreliable data from this sensor. Instead of considering an implementation of another absolute positioning technology which would be an added cost, the GPS technology that was already used for this project was studied as a possible method to provide correct heading.

Background information for the GPS technology was discussed in the earlier sections. As presented earlier, GPS heading is a parameter embedded in the NMEA statements, also known as *Track made good in degrees True*. This parameter provides a bearing value from the true north of the earth. The EM-406A module outputs a heading value every second by constantly recalculating its position with respect to the satellites that it is tracking. This update rate (1Hz) was not sufficient as events can happen within a one second interval in this Mario Kart Game. Therefore, the gyroscope was used to
measure changes of heading every 200 milliseconds. As the system takes GPS heading as the trusted heading, heading reference is updated once every second ensuring that the gyroscopes error doesn’t grow out of bounds. To complete the dead-reckoning system, displacement is measured with the optical shaft wheel encoding sensor every 200 milliseconds. Based on this data, coordinate points (Latitude and Longitude) were created using scripts written in python programming language.

By combining GPS heading with DR information, this implementation allows us to have more resolution due to its update rate, and better accuracy since it’s spanned across multiple sensors compared to GPS only implementation discussed in section 3.1.

Section 3.5 – Beacons, GPS Heading with DR

Even though GPS heading is taken as an absolute positioning method, the system computes its heading using the heading updates relative to its previous heading. Therefore, any minor could still grow out of bounds, but slowly. To prevent this, and implementation of localization with beacons paired with the GPS heading and DR was studied.

Each beacon represents a known location on the race track. These beacons would have a unique id so that the localization system of each kart would reset to the locations of these beacons every time they pass one. Only simulated beacons are implemented in this project as designing hardware for these beacons are beyond the scope of this project. The goal of using beacons was to study the effects of it on the localization system of the kart. For that purpose, a software implementation for beacons would suffice. To determine where to place beacons, certain points of interest on the race track were
measured manually and were coded in to the system such that pressing a button on the GUI (Graphical User Interface) as the user passes by the location, will initialize the localization system to the corresponding location of the beacon. The following shows a version of the GUI that was used for the beacons,

![GUI with buttons for beacons](image.png)

**Figure 23: GUI with buttons for beacons**

There are few technologies that already exist, that could be used to implement these beacons with hardware and electronics. Inductive loop detection and RFID (Radio Frequency Identification) are some of these technologies. The hardware implementation was left as future work for a possible senior project.

**Section 3.6 – Data flow and Data processing**

Information regarding the sensors that are used in creating this localization system was presented in the previous sections. These sensors use various communication protocols to communicate with their host device. A regular computer would not be able to directly communicate with these sensors since they don’t have a standard interface in common. For this reason, the Arduino microcontroller was used as a bridge between
these sensors and the host computer. Arduino microcontrollers are very low cost, well documented devices that are very popular in do-it-yourself projects. To keep the project simple and low cost, an FPGA (Field-programmable gate array) or other CPLD (Complex programmable logic device) was not used in this project. The Arduino Uno microcontroller equipped with the ATmega328 microprocessor was used for this project. The following figure shows this microcontroller,

![Arduino UNO Microcontroller](image)

**Figure 24: Arduino UNO Microcontroller [18]**

This microcontroller is capable with communicating with all communication protocols, UART, I2C, and SPI that were used in this project. The following summarizes the sensors that are connected to this microcontroller,

- The EM-406A GPS receiver uses UART protocol
- The Melexis 90609 Gyroscope uses SPI protocol
- The Honeywell HMC6352 Digital Compass uses I2C protocol
- The sensors used for the two types of wheel encoding methods send digital signals which are read by the microcontroller

This microcontroller is also equipped with USB (Universal Serial Bus), a standard communication protocol/interface that is used in almost every computer that exists today. The host computer is connected to the USB port of the microcontroller to obtain the data that were gathered from the sensors.
Upon obtaining this data, the host computer processes them according to the
scripts written in the python programming language. The main objective of these scripts
is to use the incoming data from the sensors, and create coordinates (Latitude and
Longitude). Upon creating these coordinates, they are stored along with the vital sensor
data that were used to create them. These coordinates could then be sent to the Game
Server to let it know the current position of the Karts on the race track. Due to the
existence of many test cases in this project, using a scripting language offers the
flexibility of creating code for each test case with their parameters and simple
implementation of them. Python programming language requires no compiling of the
code. It involves a simple execute command and the process from there on is automated,
unless a user input is required. Finally, for visualization purposes, the generated
coordinates were plotted on Google Earth software. The following figure shows the data
flow of the system,

![Data flow diagram](image)

Figure 25: Data flow diagram
Chapter 4 – Test Results and Analysis

This chapter presents the testing that was done on the localization methods that were implemented for this project along with their results. The track that the testing was done is located on the Dexter Lawn of the California Polytechnic State University, San Luis Obispo. The arrow on Figure 22 marks the starting point for each test run. From that point on, the kart was driven closest to the edge of grass around the track back to the starting point. The first few tests of the following test plan were not done in a chronological order. The objective of these tests was to identify the most suitable configuration of sensors for this project. After finding this configuration, the focus was shifted to increasing its accuracy. Sections 4.2 through 4.8 present the tests and their results for visual comparison followed by a brief analysis. Section 4.9 introduces the error metric used for quantitative comparison of the test results, and Section 4.10 presents the results of the quantitative comparison.

Section 4.1 – Test Plan

Based on the sensors used in this system, it was decided that the following test plan was suitable for this project.

1. GPS Only
2. Magnets + Hall-effect + Compass
3. Optical Wheel Encoder + Compass
4. Optical Wheel Encoder + Gyro
5. Optical Wheel Encoder + Kalman Filter (Compass + Gyro)
6. Optical Wheel Encoder + GPS Heading + Gyro
7. Optical Wheel Encoder + GPS Heading + Gyro + Beacons
   - Equal distance beacons (up to 10 beacons)
   - Most suitable configuration of beacons based on the observations so far
Section 4.2 - Results of the GPS only implementation

The simplest implementation of all was to obtain the latitude and the longitude parameters from the GPS receiver. Once the microcontroller was properly configured to communicate with the GPS receiver, the latitude and longitude values were extracted from the NMEA statements and were stored and displayed. The following image shows five test runs of this implementation around a chosen race track.

Test Results

Each of the green markers presented here corresponds to coordinate given by the GPS receiver, and between each of these green markers is a one second time span since the GPS receiver only has an update rate of 1Hz. Each of the yellow lines presents a test run.

Figure 26: Results of the GPS only method that didn’t provide sufficient accuracy
Analysis

The accuracy of the coordinates provided by the GPS receiver depends on many things. Among them, the number of satellites the receiver is tracking, and the number of obstructions (buildings, trees) that surrounds the race track and how close they are to the track seemed to have a major impact on the accuracy. The effects of this could be seen on the top left corner of the image above as that location is the closest to any building near this track.

The starting position for these tests was approximately at edge of the grass area where the arrow is pointing. Due to the movement of the satellites, the GPS readings are erratic. Some of the test runs are above the grass and some are beyond the race track. It can be seen that the latitude and longitude coordinates alone doesn’t have the accuracy needed for this project.

Section 4.3 – Results of Hall-effect sensor plus Magnets for wheel encoding and Digital Compass for heading (a Dead-Reckoning implementation)

The following implements the first wheel encoding method that was discussed in section 3.3. Same as the testing done for the GPS only implementation, each test run was started at the same location, and ended at the same location. The Kart was driven along the edge of the grass area. The following figure shows the test results of five test runs. Each of the blue markers represent coordinates created using the sensor data. The reason why these blue markers are placed closed together than the green ones in the figure above because in this implementation the system is updated at a higher rate (at 5 Hz).
Test Results

Figure 27: Results of Hall-effect encoding and Compass heading method that had inaccuracies in distance and heading measurements

Analysis

The issue here is that both distance and heading metrics are extremely erroneous. The total distance measurements fell short of the perimeter of the track, approximately 167 meters, by approximately 50 meters. This agrees with the data presented in section 3.3. Inaccurate distance measurements combined with inaccurate heading values provided results that are completely off the track. The above results are with the placement of the digital compass in front part of the car. To show the importance of the
placement of sensors, the following image is provided. This was when the compass was placed at the back of the car, approximately a foot away from the electric motor.

Figure 28: Results with rear mounted compass
In this case, the test run was done around the same track. However, the system plots the test runs as what’s shown above. This is due to the magnetic interference on the digital compass caused by the electric motor causing it to give incorrect heading data.

**Section 4.4 – Optical Shaft Wheel Encoding and Digital Compass for heading (A Dead-Reckoning implementation)**

In this test, the previous wheel encoding method was replaced by optical wheel encoding which aided significantly in measuring an accurate displacement.

**Test Results**

![Figure 29: Results of Optical wheel encoding and Compass heading method that provided accurate distance measurements but wrong headings](image.png)
Analysis

With this wheel encoding implementation, the total distance measured from the test results only fell short of about 5 meters to the actual distance measured around the race track. This suggested that in order to go forward with a dead-reckoning implementation, Optical Shaft wheel encoding would be the most suitable. It also suggested that the digital compass is not suitable for this project.

Section 4.5 – Optical Shaft Wheel Encoding and Gyroscope for heading (A Dead-Reckoning implementation)

In this implementation, the less accurate Digital Compass was replaced by the gyroscope to provide heading information to the system. Same test was performed and following are the results,

Test Results

Figure 30: Results of optical wheel encoding and gyroscope heading method that shows inaccuracies in heading measurements
Analysis

Since gyroscopes are not impacted by such factors as magnetic interference, their error is more predictable. As mentioned in the previous chapter, a gyroscope is used in measuring relative angular displacement. As a result, their error accumulates over time unless the heading is corrected by some other method. That is one reason for the above results. The other reason is that due to the absence of a solid 5V power rail, the sensor response was not linear. Software fixes were done to minimize the effect of this. However, this along with vibrations transferred from the terrain to the sensor are what suspected to have caused the above test results.

Section 4.6 – Optical Shaft Wheel Encoding and Kalman Filter implementation for heading (A Dead-Reckoning implementation)

In this implementation, the digital compass was given one last chance in the form of combining it with the output of the gyroscope using a Kalman filter. This filter is a two state Kalman filter where the primary state is angle/heading, and the secondary state is the rate of angular displacement. When the gyroscope doesn’t output any values or outputs low values, the filter converges to the compass value as it outputs the values that correspond to the primary state of the filter.
Test Results

Figure 31: Results of Optical wheel encoding and Kalman Filter heading method that shows an inaccurate heading output from the filter

Analysis

No noticeable improvement was seen from this implementation as the compass still outputs an inaccurate angle due to the magnetic interference. Filter was adjusted to be biased more towards the gyroscope output. However, since the gyroscope outputs an angular displacement only when movement is detected, the filter converges to the output of the compass.

Section 4.7 – GPS Heading and Dead-Reckoning Implementation

From the test results presented so far, it could be seen that the digital compass has to be replaced by some other method to measure heading. Since the gyroscope is a
relative positioning sensor that already exists in system, it would be best if an absolute positioning method was used to measure heading. To keep the cost low, GPS technology was used measure heading as it has been used in this project already. Preliminary tests showed that GPS heading when combined with GPS measurement of vehicle speed, gives a better estimate of location than the GPS coordinates alone. These preliminary tests were performed by comparing the data between the GPS coordinates and GPS heading combined GPS measurement of vehicle speed, while walking around the track. As these were preliminary results, they are not included in this thesis.

To get a better estimate of displacement, GPS measurement of vehicle speed was replaced by the more accurate Optical Shaft Wheel encoding. Also, since GPS data are updated only once every second (1Hz), gyroscope was used to update the heading in between the one second intervals. This way, the gyroscope error would not accumulate for long periods.

Due to the slower update rate (1Hz) of the GPS data, a Kalman filter was not implemented to combine the heading information from the GPS with the heading information from the Gyroscope as the heading output from the filter would potentially take longer to converge to an accurate heading.

The following figure shows the results of the GPS heading and Dead-Reckoning implementation.
Test Results

Figure 32: Results of the GPS heading and Dead-Reckoning implementation that provided the desirable accuracies

Analysis

As observed from the figure above, this implementation so far results in the best performance among the implemented localization methods. Even though this method consists of an absolute positioning method, it is still an implementation of relative positioning as the new location is updated based on the previous location. For this reason, the overall error still accumulates, even though it is not as much as the other methods implemented for this project. By making sure that the gyroscope is in its linear response range, fine tuning the wheel encoder, as well as placing the GPS receiver such that it as the maximum visibility to the sky, these errors could be minimized.
Section 4.8 – GPS Heading and Dead-Reckoning Implementation with Beacons

By visual inspection, it could be seen that the implementation in section 4.7 was the best observed so far. This implementation’s performance was studied with the introduction of beacons. These beacons would have a unique id so that the system could identify them and decode their location. By resetting the system to the location of the beacon, the overall system error is contained within bounds.

The beacons that are implemented in this thesis are software beacons as described in section 3.5. The locations of beacons were manually measured and as the user drives pass those locations, the buttons that correspond to those locations were presses on the GUI.

Beacons were placed in two different configurations. The first configuration was that beacons were placed at equal distances from each other. For the second configuration, the system response to a non-beaconed system was studied. Once the weaknesses of the system were identified, the beacons were placed to overcome these weaknesses and to maximize the accuracy of the system. A disadvantage of implementing the second configuration would be that system response to a certain track has to be studied prior to the placement of the beacons because the performance of the system varies with the track. But a smart placement of the beacons could reduce the number of beacons needed, hence keeping the cost for this localization system to a minimum.

The following figure shows the effects on the localization system by using two and three beacons that are separated by equal distances,
Test Results

Figure 33: Results with two equally spaced beacons

Figure 34: Results with three equally spaced beacons
The following figure shows the effects on the localization system by using three beacons that were placed according to the second beacon configuration (based on the observations of the system response),

![Figure 35: Results from second configuration of the beacons that provided high accuracy with low number of beacons](image)

**Analysis**

Though only three figures are shown above, the effects of the first configuration (equally spaced placement of beacons) were performed for 2 through 10 equally spaced beacons. It is important to note that the system performs less in turns than on straight paths. This is due to the accumulation of error by adding the angular displacement over time. To correct this error, beacons were placed a short distance (approximately 3 meters) after each major turn. The effects of the second configuration (smart placement of beacons) were performed for 3 and 4 beacons as there are only 4 major turns on the track that was chosen for this study.
Section 4.9 – Error Metric for Comparison of Results

In the previous sections of this chapter, figures of the results from different implementations were provided for visual comparison. To compare these results quantitatively, this section introduces an error metric.

Establishing an Ideal Path

To compare the results of each implementation, the difference between each result and an ideal result needs to be calculated. In order to obtain an ideal result, the track was surveyed to measure the angles of each corner of the track. Then, the go-kart with the best wheel encoding method (Optical Shaft Wheel Encoding) was pushed along the edge of the grass to measure distance of each side of the track. Doing so minimizes the error created by wheel slippage and sagging of the wheel under the weight of the passenger. The following figure shows the established path,

Figure 36: The Ideal Path
**Error Metric**

The results from each implementation were compared against the ideal path established above. According to [19], the error at the \(i\)th position \(\hat{X}_i\) is defined as,

\[
\hat{X}_i = \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}
\]

where \(\hat{X}_i = (x_i, y_i)\) is the true location, and \(\hat{X}_i = (\hat{x}_i, \hat{y}_i)\) is the estimated location.

Relating this to the Mario Kart project, the true location would be a location on the established ideal path, and the estimated location is a location given by the test results. However, this error metric stated in [19] defines a measurement of error pertaining to one location. To compare a series of locations on an estimated path to the ideal path, an error metric that considers the whole collection of data, is needed. For this reason Root Mean Squared Error (RMSE), was used. The following describes the RMSE,

\[
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} E_{ix}^2 + E_{iy}^2}
\]

**Figure 37: Equation for RMS Error**

RMSE is a single number that expresses the 2D accuracy. To compute the RMSE, the horizontal (along the surface of the earth) position errors along the coordinate axes (latitude and longitude) are measured and taken the square of. The sum of squared error for all locations in the path is then divided by the number of locations in the path and taken the square root of. The result of this process is easily interpreted as it has the same units as a measurement of distance [20].
Section 4.10 – Quantitative Comparison of the Results

The error metric defined in the previous section was used to qualitatively compare the results of the different implementations. To obtain the RMSE value for each implementation, calculations were performed on an average result from each method of implementation. Each of the locations of the results was compared to the corresponding estimated location on the ideal path. Presented below is a bar graph and a table of the RMSE values of the different implementations that was used to create the graph,

Table 6: RMSE values of different implementations

<table>
<thead>
<tr>
<th>Method</th>
<th>RMS Error (meters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GPS Only</td>
<td>5.540771165</td>
</tr>
<tr>
<td>2 Hall-effect encoding and Compass</td>
<td>12.25695836</td>
</tr>
<tr>
<td>3 Optical encoding and Compass</td>
<td>14.91646923</td>
</tr>
<tr>
<td>4 Optical encoding and Gyroscope</td>
<td>9.982071006</td>
</tr>
<tr>
<td>5 Optical encoding and Kalman Filter forHeading</td>
<td>11.81743757</td>
</tr>
<tr>
<td>6 Optical encoding, GPS Heading and Gyroscope</td>
<td>3.977486241</td>
</tr>
</tbody>
</table>

Figure 38: RMSE values of different implementations
An RMSE value of 0 would be ideal, but such is very difficult to achieve since every method of localization has some error associated with it. The lower the RMSE value, the closer the system response becomes to the ideal path shown in Figure 32. Based on the values in shown on Figure 33, the optical shaft wheel encoding with GPS and Gyroscope to provide heading yields the best performance of all methods studied in this thesis. To increase the accuracy of this method even more, Beacons were introduced to this implementation. As discussed in section 4.8, these beacons were implemented through software. First, the beacons were placed at equal distances from each other. The following table and graph show the RMSE values when up to 10 beacons were used to aid the localization of the method discussed in section 4.7.

<table>
<thead>
<tr>
<th>No. of Beacons</th>
<th>RMS Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.977486241</td>
</tr>
<tr>
<td>2</td>
<td>3.687506477</td>
</tr>
<tr>
<td>3</td>
<td>2.59544307</td>
</tr>
<tr>
<td>4</td>
<td>3.222072195</td>
</tr>
<tr>
<td>5</td>
<td>2.03790258</td>
</tr>
<tr>
<td>6</td>
<td>2.424589937</td>
</tr>
<tr>
<td>7</td>
<td>2.213018499</td>
</tr>
<tr>
<td>8</td>
<td>1.764495615</td>
</tr>
<tr>
<td>9</td>
<td>1.690282478</td>
</tr>
<tr>
<td>10</td>
<td>1.442219048</td>
</tr>
</tbody>
</table>

Table 7: RMSE values of beacons placed at equal distances
The local minimums of the graph shown in Figure 34 correspond to when the layout resulted beacons to be positioned closer to the exit of the major turns. As the system suffers from error in measuring angular displacement, beacons are much helpful to overcome this error when they’re placed after major turns on the track. Also observed on Figure 34 is that as the number for beacons increases, the accuracy increases. However, to keep the cost of the system lower, the least number of beacons that yields the highest accuracy should be used. Following are RMSE values of 3 and 4 beacons placed on the studied track; each beacon placed approximately 3 meters after the exit of each turn. It is very difficult to maneuver these karts since they lack an electronic or hydraulic system to aid the steering. Therefore, placing beacons any closer to the exit of the turn creates the situation where the karts never reach the spot where the beacon is placed. Three meters would give sufficient distance navigate the kart to the spot of the beacon after making the turn.

Figure 39: RMSE values of beacons placed at equal distances
Table 8: RMSE values from the second configuration of beacons

<table>
<thead>
<tr>
<th>No. of Beacons</th>
<th>RMS Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Beacons</td>
<td>3.977486241</td>
</tr>
<tr>
<td>3 Beacons</td>
<td>2.12275463865</td>
</tr>
<tr>
<td>4 Beacons</td>
<td>1.92074853168</td>
</tr>
</tbody>
</table>

The values above show that using 3 beacons placed right after each turn would approximately have the same effect as 5 equally spaced beacons, and placing 4 beacons in this manner has approximately the same effect as 7 equally spaced beacons.
Chapter 5 – Conclusion and Future Work

The primary objective of this thesis was to analyze different methods to perform localization and navigation on low speed, low cost go-karts for the Augmented Reality Mario Kart game application. Along with this objective, it was also important that the resulting localization system of this thesis is a low cost system. Chapter 2 of this thesis considers a wide range of localization methods and systems, and reasons why the chosen system is suitable for this project. The technologies that were chosen to create this system were the Global Positioning System (GPS), and Dead-Reckoning. Chapter 3 presents the different variations of the implementation. Since there are various ways to implement Dead-Reckoning, investigating which sensors to be used is essential in creating the optimum solution. Among the sensors that were used for this project, the GPS module was proven to be most useful as it provided both location and heading data. Chapter 4 presents the results of the various implementations. Google earth was used to display these result for easy visual comparison. A Quantitative analysis was also performed using the Root Mean Square Error criterion. The implementation that yielded the best results was the system that used Optical Shaft Wheel Encoding for measuring the displacement and a Gyroscope and GPS heading for measuring direction of movement. This system achieved accuracy better than GPS by itself. And also, the GPS module and sensors used for this implementation are very low in cost. Therefore, the resulting localization system successfully achieved both of its goals.

There are potential improvements that could be made for this localization system as well as the Mario Kart game system in general. The improvements that could be made for the localization system are,
1. Develop hardware for beacons

2. Develop circuitry to power sensors and microcontroller directly from the battery of the go-kart

Beacons developed in software were used to improve the accuracy of the localization system. The hardware development of the beacons was not done because it was beyond the scope of this thesis. With the aid of the beacons, the accuracy of the system was significantly improved. Therefore, it would be very useful if a low cost hardware implementation of the beacons is developed in the future.

Due to the absence of a solid 5 volt supply line, some sensors failed to perform in their normal response range. There sensors had to be configured through software so that they would respond normally to the given power supply line. It would be very convenient and more accurate if a 5 volt supply line is derived directly from the battery of the go-kart. To achieve this, a small DC-to-DC regulator such as the following from linear technology, can be used.

![Diagram of 5V/2A Buck Converter](image-url)

Figure 40: LTC3824 controller for Linear Technology [21]
Efficiency and the quiescent current of the regulator are a very important factor in choosing a regulator for this project. These go-karts run on 36 Volt rechargeable batteries that are capable of supplying limited amount of energy per charger. The main battery has to supply power to the motor that drives the kart, the motor controller, and the electronics needed for the localization system. For the periods when the kart is idle, the motor and the controller are not powered, but the electronics are. Therefore, it is important to have a high efficient DC-to-DC regulator to dissipate the least amount of energy from the main battery. This DC-to-DC regulator should come at a low cost as it is one of the main objectives of this project to create low cost solutions. The total cost of parts for the above circuit as quoted by Digi-Key Corporation is $14.39 at the time of this writing. This quotation is attached in the appendix. The other important factor in designing this regulator circuit is that this circuit has to output a solid 5 volt. As experienced in this project, any output other than this voltage is going to contribute towards error in sensor readings. Therefore, careful consideration of feedback resistors is essential.

With the addition of beacons, and a separate power supply for the electronics used for localization, the accuracy and the reliability of the localization system could be improved.
Bibliography


[Accessed 15 7 2012].


Appendix A – Microcontroller Code

The following code was written in C programming language for the Arduino Microcontroller.

A.1 – Code to communicate with the USGlobalSat EM-406A GPS Receiver

```c
byte GPS_Rx;
void setup()
{
  Serial.begin(4800);
}

void loop()
{
  if (Serial.available()>0)
  {
    GPS_Rx = Serial.read();
    Serial.print(GPS_Rx, BYTE);
  }
}
```

A.2 – Code to communicate with the US1881 Hall-effect sensor and HMC6352 Digital Compass

The following code is based on the implementation done in [22].

```c
#include <Wire.h>
#include "TimerOne.h"
#include <digitalWriteFast.h>

/* WHEEL ENCODER */
#define EncoderPIN 2 // digital interrupt pin 0
#define CIRCUMFERENCE 0.166504411
#define TICKS 6 //number of ticks on the wheel = 1 revolution
#define TIME 1
#define THRESHOLD 5
#define V_SCALE 2
```
/* WHEEL ENCODER */
long int count, oldCount, newCount, countDiff;
float velocity;
float disTravel = 0.0;

// Compass
#include <Wire.h>
int HMC6352Address = 0x42;
// This is calculated in the setup() function
int slaveAddress;
byte headingData[2];
int i, headingValue;

void setup()
{
    Timer1.initialize(200000);
    slaveAddress = HMC6352Address >> 1;   // This results in 0x21
    as the address to pass to TWI
    Wire.begin();
    pinMode(EncoderPIN, INPUT);
    attachInterrupt(0, encoderEvent, CHANGE);
    Serial.begin(115200);
    Timer1.attachInterrupt(callback);
}

void loop()
{
    // Send a "A" command to the HMC6352
    // This requests the current heading data
    Wire.beginTransmission(slaveAddress);
    Wire.send("A");              // The "Get Data" command
    Wire.endTransmission();
    delay(10);                     // The HMC6352 needs at least a
    70us (microsecond) delay
    // after this command. Using 10ms just makes it safe
    // Read the 2 heading bytes, MSB first
    // The resulting 16bit word is the compass heading in 10th's of
    a degree
    // For example: a heading of 1345 would be 134.5 degrees
    Wire.requestFrom(slaveAddress, 2);    // Request the 2 byte
    heading (MSB comes first)
    i = 0;
    while(Wire.available() && i < 2)
    {

headingData[i] = Wire.receive();
i++;
}
headingValue = headingData[0]*256 + headingData[1]; // Put the
MSB and LSB together
delay(100);

/* Event to happen during change from HI - LOW or vice versa */
void encoderEvent()
{
    count++;
}

/* Outputs the velocity of the cart in .1 meters per second */
void wheelEncoder()
{
    newCount = count;
countDiff = newCount-oldCount;
oldCount = newCount;

    float ratio = (float)countDiff/(float)TICKS;
velocity = ratio/(float)TIME*CIRCUMFERENCE;
if(velocity > THRESHOLD) velocity = 2;
velocity = velocity * V_SCALE;
// Serial.println(velocity);
}

void callback()
{
    wheelEncoder();
disTravel = disTravel + (velocity*10*0.2);
Serial.print(int (headingValue / 10)); // The whole number part of the heading
Serial.print(",");
Serial.print(int (headingValue % 10)); // The fractional part of the heading
Serial.print(",");
Serial.println(disTravel);
}
A.3 – Code to communicate with Optical Shaft Wheel Encoder and the MLX90609 Gyroscope

The wheel encoder portion of the following code is based on [23], and the gyroscope portion is based on [24].

#include <Wire.h>
#include "TimerOne.h"
#include <digitalWriteFast.h>

// Quadrature encoders
// Left encoder
#define c_LeftEncoderInterrupt 0
#define c_LeftEncoderPinA 2
#define c_LeftEncoderPinB 4
volatile bool _LeftEncoderBSet;
volatile long _LeftEncoderTicks = 0;
long newTicks = 0, oldTicks = 0, diffTicks = 0;

//Gyro
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO - not used, but part of builtin SPI
#define SPICLOCK 13//sck
#define CS_GYRO 10//ss

int rate_gyro = 0;
int total_rate = 0;
int gyro_count = 0;

void setup()
{
    Timer1.initialize(200000);
    //Gyro Pins
    pinMode(DATAOUT, OUTPUT);
    pinMode(DATAIN, INPUT);
    pinMode(SPICLOCK, OUTPUT);
    pinMode(CS_GYRO, OUTPUT);
    digitalWrite(CS_GYRO, HIGH); //disable device
    //set SPI speed, not needed!?  
    //SPSR = (1<<SPI2X);
    configureSPI_GYRO();

    Serial.begin(115200);
// Quadrature encoders
// Left encoder
pinMode(c_LeftEncoderPinA, INPUT);       // sets pin A as input
//digitalWrite(c_LeftEncoderPinA, LOW);  // turn on pullup resistors
pinMode(c_LeftEncoderPinB, INPUT);       // sets pin B as input
//digitalWrite(c_LeftEncoderPinB, LOW);  // turn on pullup resistors
attachInterrupt(c_LeftEncoderInterrupt,
HandleLeftMotorInterruptA, RISING);
Timer1.attachInterrupt(callback);
}

void configureSPI_GYRO()
{
    SPCR = /* Configure SPI mode: */
        // (1<<SPIE) | /* should be activated to enable interruption from the SPI */
        (1<<SPE) | /* to enable SPI */
        (1<<MSTR) | /* to set Master SPI mode */
        (1<<CPOL) | /* SCK is high when idle */
        (1<<CPHA) | /* data is sampled on the trailing edge of the SCK */
        // (1<<SPR1) | /* In this example SPI0=1, SPR1=0 (commented) and SPI2X=1 */
        (1<<SPR0); /* It sets SCK freq. in 8 times less than a system clock */
}

/*
generic SPI read/write function, can be used with any device */
char spi_transfer(unsigned char data)
{
    SPDR = data; // Start the transmission
    while (!(SPSR & (1<<SPIF))){}; // Wait the end of the transmission
    return SPDR; // return the received byte
}

unsigned int getAngularRate()
{
    byte data1, data2;

digitalWrite(CS_GYRO, LOW);
spi_transfer(0b10010100);
digitalWrite(CS_GYRO,HIGH);

delayMicroseconds(250);

digitalWrite(CS_GYRO,LOW);
spi_transfer(0b10000000);  // Send SPI ADCR Instruction
data1 = spi_transfer(0x00);
data2 = spi_transfer(0x00);
digitalWrite(CS_GYRO,HIGH);

unsigned int result = ((data1 & B00001111) << 7) + (data2>>1);
return result;
}

int toAngularRate(unsigned int adcValue)
{
    int conversion = (adcValue * 25/12)+400-34;
    if (conversion > 4458){
        return 0;
    }
    else{
        signed int offset = 2431;
        return (conversion - offset)/30.67;
    }
}

void loop()
{
    rate_gyro = toAngularRate(getAngularRate());
total_rate = total_rate + rate_gyro;
gyro_count = gyro_count + 1;
delay(10);
}

// Interrupt service routines for the left motor's quadrature encoder
void HandleLeftMotorInterruptA()
{
    // Test transition; since the interrupt will only fire on 'rising' we don't need to read pin A
    _LeftEncoderBSet = digitalReadFast(c_LeftEncoderPinB);  // read the input pin

    // and adjust counter + if A leads B
#ifdef LeftEncoderIsReversed
 _LeftEncoderTicks -= _LeftEncoderBSet ? -1 : +1;
#else
 _LeftEncoderTicks += _LeftEncoderBSet ? -1 : +1;
#endif

void callback()
{
  //try this
  int Avg_rate = total_rate/gyro_count;
  gyro_count = 0;
  total_rate = 0;

  Serial.print(Avg_rate);
  Serial.print(",");
  Serial.println(_LeftEncoderTicks);
}

A.4 – Code for implementation of the Kalman Filter

The following implementation of the Kalman Filter is based on [17], as discussed in Section 3.3.2.

#include <math.h>
#include <Wire.h>
#include "TimerOne.h"
#include <digitalWriteFast.h>

// Quadrature encoders
// Left encoder
#define c_LeftEncoderInterrupt 0
#define c_LeftEncoderPinA 2
#define c_LeftEncoderPinB 4
volatile bool _LeftEncoderBSet;
volatile long _LeftEncoderTicks = 0;
long newTicks = 0, oldTicks = 0, diffTicks = 0;

//Gyro
#define DATAOUT 11//MOSI
#define DATAIN 12//MISO – not used, but part of builtin SPI
#define SPICLOCK 13//sck
#define CS_GYRO 10//ss
// Compass
int HMC6352Address = 0x42;
// This is calculated in the setup() function
int slaveAddress;
byte headingData[2];
int i, headingValue;

static const float dt = 0.16;

static float P[2][2] = {
    { 1, 0 },
    { 0, 1 },
};

float angle;
float rate;
float q_bias;

int rate_gyro = 0;
int total_rate = 0;
int gyro_count = 0;

// R represents the measurement covariance noise
static const float R_angle = 0.3;

// Q is a 2x2 matrix that represents the process covariance noise
static const float Q_angle = 0.001;
static const float Q_gyro = 0.005;

void setup()
{
    Timer1.initialize(200000);
    slaveAddress = HMC6352Address >> 1;   // This results in 0x21 as the address to pass to TWI
    Wire.begin();
    // Gyro Pins
    pinMode(DATAOUT, OUTPUT);
    pinMode(DATAIN, INPUT);
    pinMode(SPILOCK, OUTPUT);
    pinMode(CS_GYRO, OUTPUT);
    digitalWrite(CS_GYRO, HIGH); // disable device
    // set SPI speed, not needed!?
    // SPSR = (1<<SPI2X);
    configureSPI_GYRO();
}
Serial.begin(115200);
// Quadrature encoders
// Left encoder
pinMode(c_LeftEncoderPinA, INPUT);  // sets pin A as input
//digitalWrite(c_LeftEncoderPinA, LOW);  // turn on pullup resistors
pinMode(c_LeftEncoderPinB, INPUT);  // sets pin B as input
//digitalWrite(c_LeftEncoderPinB, LOW);  // turn on pullup resistors
attachInterrupt(c_LeftEncoderInterrupt,
HandleLeftMotorInterruptA, RISING);
Timer1.attachInterrupt(callback);
}

void configureSPI_GYRO()
{
    SPCR = /* Configure SPI mode: */
    // (1<<SPIE) | /* should be activated to enable interruption
    from the SPI */
    (1<<SPE) | /* to enable SPI */
    (1<<MSTR) | /* to set Master SPI mode */
    (1<<CPOL) | /* SCK is high when idle */
    (1<<CPHA) | /* data is sampled on the trailing edge of the
    SCK */
    // (1<<SPR1) | /* In this example SPI0=1, SPR1=0 (commented)
    and SPI2X=1 */
    (1<<SPR0); /* It sets SCK freq. in 8 times less than a system
    clock */
}

/*
generic SPI read/write function, can be used with any device
*/
char spi_transfer(unsigned char data)
{
    SPDR = data;  // Start the transmission
    while (!(SPSR & (1<<SPIF))){};  // Wait the end of the
    transmission
    return SPDR;  // return the received byte
}

unsigned int getAngularRate()
{
    byte data1, data2;
digitalWrite(CS_GYRO,LOW);
spi_transfer(0b10010100);
digitalWrite(CS_GYRO,HIGH);

delayMicroseconds(250);

digitalWrite(CS_GYRO,LOW);
spi_transfer(0b10000000);  // Send SPI ADCR Instruction
data1 = spi_transfer(0x00);
data2 = spi_transfer(0x00);
digitalWrite(CS_GYRO,HIGH);

unsigned int result = ((data1 & B00001111) << 7) + (data2>>1);
return result;

int toAngularRate(unsigned int adcValue)
{
    int conversion = (adcValue * 25/12)+400-34;
    if (conversion > 4458){
        return 0;
    }
    else{
        signed int offset = 2412;
        return (conversion - offset)/31.67;
    }
}

void loop()
{
    // Send a "A" command to the HMC6352
    // This requests the current heading data
    Wire.beginTransmission(slaveAddress);
    Wire.send("A");              // The "Get Data" command
    Wire.endTransmission();
    delay(10);                   // The HMC6352 needs at least a
70us (microsecond) delay
    // after this command. Using 10ms just makes it safe
    // Read the 2 heading bytes, MSB first
    // The resulting 16bit word is the compass heading in 10th's of
a degree
    // For example: a heading of 1345 would be 134.5 degrees
    Wire.requestFrom(slaveAddress, 2);        // Request the 2 byte
heading (MSB comes first)
i = 0;
while(Wire.available() && i < 2)
{
    headingData[i] = Wire.receive();
i++;
}
headingValue = headingData[0]*256 + headingData[1];  // Put the MSB and LSB together
rate_gyro = toAngularRate(getAngularRate());
total_rate = total_rate + rate_gyro;
gyro_count = gyro_count + 1;
state_update(rate_gyro);
float heading = headingValue/10;
kalman_update(heading);
delay(100);//was 10
}

void HandleLeftMotorInterruptA()
{
    // Test transition; since the interrupt will only fire on 'rising' we don't need to read pin A
    _LeftEncoderBSet = digitalReadFast(c_LeftEncoderPinB);   // read the input pin

    // and adjust counter + if A leads B
    #ifdef LeftEncoderIsReversed
        _LeftEncoderTicks -= _LeftEncoderBSet ? -1 : +1;
    #else
        _LeftEncoderTicks += _LeftEncoderBSet ? -1 : +1;
    #endif
}

void callback()
{
    int Avg_rate = total_rate/gyro_count;
gyro_count = 0;
total_rate = 0;

    Serial.print(int (headingValue / 10));     // The whole number part of the heading
    Serial.print(".");
    Serial.print(int (headingValue % 10));     // The fractional part of the heading
    Serial.print(",");
    Serial.print(Avg_rate);
}
void state_update(const float q_m){
    const float q = q_m - q_bias;
    const float Pdot[2 * 2] = {
        Q_angle - P[0][1] - P[1][0],
        - P[1][1],
        - P[1][1],
        Q_gyro
    };
    rate = q;
    angle += q * dt;
    //Update the covariance matrix
    P[0][0] += Pdot[0] * dt;
    P[0][1] += Pdot[1] * dt;
    P[1][0] += Pdot[2] * dt;
}

void kalman_update(const float angle_m){
    const float angle_err = angle_m - angle;
    const float C_0 = 1;
    const float PCT_0 = C_0 * P[0][0];
    const float PCT_1 = C_0 * P[1][0];
    const float E = R_angle + C_0 * PCT_0;
    const float K_0 = PCT_0 / E;
    const float K_1 = PCT_1 / E;
    const float t_0 = PCT_0;
    const float t_1 = C_0 * P[0][1];
P[0][0] -= K_0 * t_0;
P[0][1] -= K_0 * t_1;
P[1][0] -= K_1 * t_0;
P[1][1] -= K_1 * t_1;

angle += K_0 * angle_err;
q_bias += K_1 * angle_err;
}
Appendix B – Test scripts written in Python Programming Language

The following Test scripts were written in Python to generate coordinates (Latitude and Longitude) based on the data that the host computer received from the Arduino microcontroller.

B.1 – Code for generating new coordinates

```python
#This algorithm should find the coordinates of a destination point
# given the coordinates of the starting point, bearing, and speed
#Inputs:- coordinates of the starting point (in degrees), bearing (in degrees),
# displacement (in cm)
#Outputs:- coordinates of the destination point (in degrees)

import math as math

#Constants
#Earth's radius 6371km
EARTH_RADIUS = 6371

class LatLonCalc():
    def __init__(self, parent=None):
        index = 0

        #def find_distance(self, speed, sampling_interval):
        # 1 knot = 1.85200 km
        #distance = (speed*1.85200*sampling_interval)/3600

        #return distance

        def Calculate_New_coor(self, lat1, lon1, displacement, bearing):
            #Calculate the distance travelled in km
            self.distance = (displacement * 10e-6)

            self.d_R = self.distance/EARTH_RADIUS
```

76
#Converting everything into radians
self.latitude1 = math.radians(lat1)
self.longitude1 = math.radians(lon1)
self.bearing1 = math.radians(bearing)

self.latitude2 =
math.asin((math.sin(self.latitude1)*math.cos(self.d_R)) +
(math.cos(self.latitude1)*math.sin(self.d_R)*math.cos(self.bearing1)))

self.lon2 = self.longitude1 +
math.atan2((math.sin(self.bearing1)*math.sin(self.d_R)*math.cos(self.latitude1)),(math.cos(self.d_R) -
(math.sin(self.latitude1)*math.sin(self.latitude2))))

#Normalize to -180 to +180
self.longitude2 = ((self.lon2 + math.pi)%(2*math.pi)) - math.pi

#Converting it back to degrees
self.longitude2 = math.degrees(self.longitude2)
self.latitude2 = math.degrees(self.latitude2)
return self.latitude2, self.longitude2

B.2 – Code for GPS heading and Dead-Reckoning implementation

The following code also imports gpssparser by [25] as a module to decode the NMEA strings received from the USGlobalSat EM-406A GPS receiver.

import gpssparser as gps
import GPSTest_GUI as GUI
import CalcLat2Lon2 as CalcNextCoor
import CalcLat2Lon2GPS as CalcNextCoorGPS
from PyQt4.QtCore import *
from PyQt4.QtGui import *

#getting the GPS data from the COM port and determining if its viable
import serial as communi
import sys
import os
from datetime import datetime

app = QApplication(sys.argv)
GUI2 = GUI.GPSForm()
NextCoor = CalcNextCoor.LatLonCalc()
NextCoorGPS = CalcNextCoorGPS.LatLonCalc()
coor = gps.GPSString('')
timer_GPS = QTimer()
timer_Kart = QTimer()

# Creating a folder for the data to be logged
date_time = str(datetime.now())
data_folder = 'GPSHeadingPlusGyro Data ' + date_time[0:13] + "-" + date_time[14:16] + "-" + date_time[17:19]
folder_location = '/Users/MarioTop 2/Documents/Arduino/Test Cases/OpticalEncoderPlus(GPS+Gyro)/Python/Data Log Files/' + data_folder
os.mkdir(folder_location)

GPSData_file_location = folder_location + '/' + 'GPSData.txt'
CalcLatLonGPS_file_location = folder_location + '/' + 'CalculatedGPS_LatLon.txt'
CalcLatLon_file_location = folder_location + '/' + 'Calculated_LatLon.txt'
kartdata_file_location = folder_location + '/' + 'KartData.txt'

GPSData_file = open(GPSData_file_location, "wb")
CalcLatLon_file = open(CalcLatLon_file_location, "wb")
CalcLatLonGPS_file = open(CalcLatLonGPS_file_location, "wb")
kartdata_file = open(kartdata_file_location, "wb")

# Change the port number if needed
# GPS
locdata_GPS = communi.Serial(port=5, baudrate=4800,
bytesize=communi.EIGHTBITS, parity=communi.PARITY_NONE,
stopbits=communi.STOPBITS_ONE, timeout=None, xonxoff=False,
rtscts=False, dsrdtr=False, writeTimeout=False,
interCharTimeout=None)
# Kart Electronics
locdata_Kart = communi.Serial(port=4, baudrate=115200,
bytesize=communi.EIGHTBITS, parity=communi.PARITY_NONE,
stopbits=communi.STOPBITS_ONE, timeout=None, xonxoff=False,
rtscts=False, dsrdtr=False, writeTimeout=False,
interCharTimeout=None)
locdata_GPS.close()
locdata_Kart.close()

lat = 0.0
lat_GPS = 0.0
lon = 0.0
lon_GPS = 0.0
oldTickVal = 0.0
GPS_bearing = 0.0
count_GPS_bearing = 0

def initial_read_gps_and_compass():
    global GPS_bearing
    global lat
    global lat_GPS
    global lon
    global lon_GPS

    ignore_gps_data = True
    gps_time = str('0')
gps_alt = str('0')
    while(ignore_gps_data == True):
        gpsdata = locdata_GPS.readline()
        print(gpsdata)
gpsdata_stripped = gpsdata.split(',';
        if (gpsdata_stripped[0] == '$GPGGA'):
               gpsdata_stripped[6] == '' or gpsdata_stripped[7] == '' or
               gpsdata_stripped[8] == '' or gpsdata_stripped[9] == '' or
               gpsdata_stripped[12] == '' or gpsdata_stripped[14] == ''):
                ignore_gps_data = True
                print('No initial position')
        else:
            ignore_gps_data = False
            print('Got initial position')
            GPSData_file.write(gpsdata)
            gps_time = str(gpsdata_stripped[1])
gps_alt = str(gpsdata_stripped[9])

        else:
            ignore_gps_data = True
lat = 35.300706
lat_GPS = 35.300706
lon = -120.663394
lon_GPS = -120.663394
GPS_bearing = float(167)
print('Got initial bearing: ' + str(GPS_bearing))
CalcLatLonGPS_file.write(gps_time + ',' + str(lon_GPS) + ',' + str(lat_GPS) + ',' + gps_alt + '
')
CalcLatLon_file.write(gps_time + ',' + str(lon) + ',' + str(lat) + ',' + gps_alt + '
')

def read_gps():
    ignore_gps_data = True
    global GPS_bearing
    global count_GPS_bearing

    while(ignore_gps_data == True):
        gpsdata = locdata_GPS.readline()
        GPSData_file.write(gpsdata)
        gpsdata_stripped = gpsdata.split(',','
')
        if (gpsdata_stripped[0] == '$GPRMC'):
            ignore_gps_data = False

            time = gpsdata_stripped[1]
speed = float(gpsdata_stripped[7])
        if (count_GPS_bearing <= 4):
            GPS_bearing = float(167)
            count_GPS_bearing = count_GPS_bearing + 1
        else:
            GPS_bearing = float(gpsdata_stripped[8])

    else:
        ignore_gps_data = True

    return time, speed, GPS_bearing

def read_KartDispBearing():
    global oldTickVal
    global GPS_bearing
    global lat
    global lon
    global NextCoor

    time_now = datetime.now()
    kartdata = locdata_Kart.readline()
print(kartdata)  
kartdata_file.write(str(time_now.time()) + ',' + kartdata)  
kartdata_stripped = kartdata.split(',')  
newTickVal = float(kartdata_stripped[1])  
newDisp = abs(newTickVal - oldTickVal)  
newDispcm = (newDisp/200)*79.545  
oldTickVal = newTickVal  
if (int(kartdata_stripped[0]) >= -1 and  
int(kartdata_stripped[0]) <= 1):  
    GPS_bearing = GPS_bearing + 0.0  
else:  
    GPS_bearing = GPS_bearing + float(kartdata_stripped[0])  
    GPS_bearing = GPS_bearing%360  
(lat, lon) = NextCoor.Calculate_New_coor(lat, lon, newDispcm,  
GPS_bearing)  
CalcLatLon_file.write(str(time_now.time()) + ',' + str(lon) +  
',' + str(lat) + '\r\n')


timer_Kart.timeout.connect(read_KartDispBearing)

def calculate_next_coor():
    global lat_GPS  
    global lon_GPS  
    global NextCoorGPS

    (gps_time, gps_speed, gps_bearing) = read_gps()  
    time_now = datetime.now()  
    print('GPS Speed = ' + str(gps_speed) + ' Bearing = ' +  
str(gps_bearing))  
    print('Old latitude = ' + str(lat_GPS))  
    print('Old longitude = ' + str(lon_GPS))  
    (lat_GPS, lon_GPS) = NextCoorGPS.Calculate_New_coor(lat_GPS,  
lon_GPS, gps_speed, gps_bearing, 0.990)  
    print('New latitude = ' + str(lat_GPS))  
    print('New longitude = ' + str(lon_GPS))  
    CalcLatLonGPS_file.write(str(time_now.time()) + ',' +  
str(lon_GPS) + ',' + str(lat_GPS) + '\r\n')

    timer_GPS.timeout.connect(calculate_next_coor)

def Initialize_Clicked():
    global locdata_GPS  
    global locdata_Kart

    print('ok')
locdata_GPS.open()
locdata_Kart.open()
initial_read_gps_and_compass()
locdata_Kart.close()
locdata_GPS.close()

def RUN_Clicked():
global GUI2
global timer_Kart
global timer_GPS
global locdata_GPS
global locdata_Kart

dlodata_Kart.open()
dlodata_GPS.open()
timer_Kart.start(180) #was 100
timer_GPS.start(990)

def STOP_Clicked():
global GUI2
global timer_Kart
global timer_GPS
global locdata_GPS
global locdata_Kart

timer_Kart.stop()
timer_GPS.stop()
locdata_Kart.close()
locdata_GPS.close()

def Close_Clicked():
global locdata_Kart
global locdata_GPS

timer_Kart.stop()
timer_GPS.stop()
locdata_Kart.close()
locdata_GPS.close()
GPSData_file.close()
CalcLatLonGPS_file.close()
CalcLatLon_file.close()
kartdata_file.close()

sys.exit()
GUI2.InitializeButton.clicked.connect(Initialize_Clicked)
GUI2.RunButton.clicked.connect(RUN_Clicked)
GUI2.StopButton.clicked.connect(STOP_Clicked)
GUI2.CloseButton.clicked.connect(Close_Clicked)
app.exec_()

B.3 – Code for the Graphical User Interfaces

Code in GPTest_GUI.py file is shown below

```python
import re
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from ui_GPSAcq import *

class GPSForm(QDialog, Ui_gpsDialog):
    def __init__(self, parent=None):
        super(GPSForm, self).__init__(parent)
        self.setupUi(self)
        self.show()

if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)
    GPS = GPSForm()
    GPS.show()
    sys.exit(app.exec_())
```

Code in ui_GPSAcq.py file is shown below

```python
from PyQt4 import QtCore, QtGui
try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s
```
class Ui_gpsDialog(object):
    def setupUi(self, gpsDialog):
        gpsDialog.setObjectName(_fromUtf8("gpsDialog"))
        gpsDialog.resize(400, 300)
        self.buttonBox = QtGui.QDialogButtonBox(gpsDialog)
        self.buttonBox.setGeometry(QtCore.QRect(30, 240, 341, 32))
        self.buttonBox.setOrientation(QtCore.Qt.Horizontal)
        self.buttonBox.setStandardButtons((QtGui.QDialogButtonBox.Cancel|QtGui.QDialogButtonBox.Ok))
        self.buttonBox.setObjectName(_fromUtf8("buttonBox"))
        self.InitializeButton = QtGui.QPushButton(gpsDialog)
        self.InitializeButton.setGeometry(QtCore.QRect(150, 40, 75, 23))
        self.InitializeButton.setObjectName(_fromUtf8("InitializeButton"))
        self.RunButton = QtGui.QPushButton(gpsDialog)
        self.RunButton.setGeometry(QtCore.QRect(150, 80, 75, 23))
        self.RunButton.setObjectName(_fromUtf8("RunButton"))
        self.StopButton = QtGui.QPushButton(gpsDialog)
        self.StopButton.setGeometry(QtCore.QRect(150, 120, 75, 23))
        self.StopButton.setObjectName(_fromUtf8("StopButton"))
        self.CloseButton = QtGui.QPushButton(gpsDialog)
        self.CloseButton.setGeometry(QtCore.QRect(150, 160, 75, 23))
        self.CloseButton.setObjectName(_fromUtf8("CloseButton"))

        self.retranslateUi(gpsDialog)
        QtCore.QObject.connect(self.buttonBox,
                                QtCore.SIGNAL(_fromUtf8("accepted()")), gpsDialog.accept)
        QtCore.QObject.connect(self.buttonBox,
                                QtCore.SIGNAL(_fromUtf8("rejected()")), gpsDialog.reject)
        QtCore.QMetaObject.connectSlotsByName(gpsDialog)

    def retranslateUi(self, gpsDialog):
        gpsDialog.setWindowTitle(QtGui.QApplication.translate("gpsDialog", "Dialog", None, QtGui.QApplication.UnicodeUTF8))
        self.InitializeButton.setText(QtGui.QApplication.translate("gpsDialog", "Initialize", None, QtGui.QApplication.UnicodeUTF8))

    def retranslateUi(self, gpsDialog):
        gpsDialog.setWindowTitle(QtGui.QApplication.translate("gpsDialog", "Dialog", None, QtGui.QApplication.UnicodeUTF8))
        self.InitializeButton.setText(QtGui.QApplication.translate("gpsDialog", "Initialize", None, QtGui.QApplication.UnicodeUTF8))
self.RunButton.setText(QtGui.QApplication.translate("gpsDialog", "Run", None, QtGui.QApplication.UnicodeUTF8))

self.StopButton.setText(QtGui.QApplication.translate("gpsDialog", "Stop", None, QtGui.QApplication.UnicodeUTF8))

self.CloseButton.setText(QtGui.QApplication.translate("gpsDialog", "Close", None, QtGui.QApplication.UnicodeUTF8))

Code in Gyro_GUI.py file is shown below

```python
import re
from PyQt4.QtCore import *
from PyQt4.QtGui import *
from ui_rate import *

accumulated_value = 0
accumulated_rotations = 0

class RateForm(QDialog,Ui_rateDialog):
    def __init__(self, parent=None):
        super(RateForm, self).__init__(parent)
        self.setupUi(self)

        self.gyrodial.setMinimum(-179)
        self.gyrodial.setMaximum(180)

        self.compassdial.setMinimum(-179)
        self.compassdial.setMaximum(180)

        self.kalmanDial.setMinimum(-179)
        self.kalmanDial.setMaximum(180)

        self.DisplayRate_gyro(0,0)
        self.DisplayRate_compass(0,0)
        self.DisplayRate_kalman(0,0)
        self.show()

    def DisplayRate_gyro(self, text,value):
        global accumulated_value
        global accumulated_rotations
```
self.lcdgyrorate.display(text)
accumulated_value = accumulated_value + value
self.lcdbearing.display(accumulated_value%360)

if (accumulated_value > 180):
    accumulated_value = accumulated_value - 360
    accumulated_rotations = accumulated_rotations + 1

self.gyro.dial.setValue(accumulated_value)
self.label_2.setNum(accumulated_rotations)

def DisplayRate_kalman(self, text, value):
    self.kalmanDial.setValue(value)
    self.lcdkalman.display(text)

def DisplayRate_compass(self, text, value):
    self.lcdcompassbearing.display(text)
    self.compassdial.setValue(value)

if __name__ == '__main__':

    import sys

    app = QApplication(sys.argv)
    Form = RateForm()
    Form.show()
    sys.exit(app.exec_())

Code in ui_rate.py file is shown below

from PyQt4 import import QtCore, QtGui

try:
    _fromUtf8 = QtCore.QString.fromUtf8
except AttributeError:
    _fromUtf8 = lambda s: s

class Ui_rateDialog(object):
    def setupUi(self, rateDialog):
        rateDialog.setObjectName(_fromUtf8("rateDialog"))
        rateDialog.resize(1241, 314)
        self.gyro.dial = QtGui.QDial(rateDialog)
self.gyrodial.setGeometry(QtCore.QRect(220, 40, 151, 161))
self.gyrodial.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))
self.gyrodial.setWrapping(True)
self.gyrodial.setNotchesVisible(True)
self.gyrodial.setObjectName(_fromUtf8("gyrodial"))
self.lcdbearing = QtGui.QLCDNumber(rateDialog)
self.lcdbearing.setGeometry(QtCore.QRect(50, 110, 121, 51))
self.lcdbearing.setObjectName(_fromUtf8("lcdbearing"))
self.label = QtGui.QLabel(rateDialog)
self.label.setGeometry(QtCore.QRect(70, 0, 111, 41))
font = QtGui.QFont()
font.setPointSize(14)
self.label.setFont(font)
self.label.setObjectName(_fromUtf8("label"))
self.label_2 = QtGui.QLabel(rateDialog)
self.label_2.setGeometry(QtCore.QRect(290, 30, 46, 13))
font = QtGui.QFont()
font.setPointSize(11)
self.label_2.setFont(font)
self.label_2.setObjectName(_fromUtf8("label_2"))
self.pushButton_Run = QtGui.QPushButton(rateDialog)
self.pushButton_Run.setGeometry(QtCore.QRect(70, 200, 75, 23))
self.pushButton_Run.setObjectName(_fromUtf8("pushButton_Run"))
self.pushButton_Stop = QtGui.QPushButton(rateDialog)
self.pushButton_Stop.setGeometry(QtCore.QRect(70, 230, 75, 23))
self.pushButton_Stop.setObjectName(_fromUtf8("pushButton_Stop"))
self.pushButton_Close = QtGui.QPushButton(rateDialog)
self.pushButton_Close.setGeometry(QtCore.QRect(70, 260, 75, 23))
self.pushButton_Close.setObjectName(_fromUtf8("pushButton_Close"))
self.lineCOM = QtGui.QLineEdit(rateDialog)
self.lineCOM.setGeometry(QtCore.QRect(320, 230, 61, 20))
self.lineCOM.setObjectName(_fromUtf8("lineCOM"))
self.label_3 = QtGui.QLabel(rateDialog)
self.label_3.setGeometry(QtCore.QRect(200, 230, 111, 21))
font = QtGui.QFont()
font.setPointSize(10)
self.label_3.setFont(font)
self.label_3.setObjectName(_fromUtf8("label_3"))
self.lcdgyrorate = QtGui.QLCDNumber(rateDialog)
self.lcdgyrorate.setGeometry(QtCore.QRect(50, 50, 121, 51))
self.lcdgyrorate.setObjectName(_fromUtf8("lcdgyrorate"))
self.pushButton_Init = QtGui.QPushButton(rateDialog)
pushButton_Init.setGeometry(QtCore.QRect(250, 260, 108, 23))
pushButton_Init.setObjectName(_fromUtf8("pushButton_Init"))
self.compassdial = QtGui.QDial(rateDialog)
compassdial.setGeometry(QtCore.QRect(600, 50, 161, 151))
compassdial.setOrientation(QtCore.Qt.Horizontal)
compassdial.setInvertedAppearance(False)
compassdial.setInvertedControls(False)
compassdial.setWrapping(True)
compassdial.setNotchesVisible(True)
compassdial.setObjectName(_fromUtf8("compassdial"))
self.lcdcompassbearing = QtGui.QLCDNumber(rateDialog)
lcdcompassbearing.setGeometry(QtCore.QRect(460, 50, 121, 51))
lcdcompassbearing.setObjectName(_fromUtf8("lcdcompassbearing"))
self.label_4 = QtGui.QLabel(rateDialog)
label_4.setGeometry(QtCore.QRect(480, 10, 121, 31))
font = QtGui.QFont()
font.setPointSize(14)
label_4.setFont(font)
label_4.setObjectName(_fromUtf8("label_4"))
self.label_5 = QtGui.QLabel(rateDialog)
label_5.setGeometry(QtCore.QRect(680, 30, 51, 21))
font = QtGui.QFont()
font.setPointSize(10)
label_5.setFont(font)
label_5.setObjectName(_fromUtf8("label_5"))
self.label_6 = QtGui.QLabel(rateDialog)
label_6.setGeometry(QtCore.QRect(570, 230, 111, 21))
font = QtGui.QFont()
font.setPointSize(10)
label_6.setFont(font)
label_6.setObjectName(_fromUtf8("label_6"))
self.lineCOM_compass = QtGui.QLineEdit(rateDialog)
    self.lineCOM_compass.setGeometry(QtCore.QRect(700, 230, 61, 20))

    self.lineCOM_compass.setObjectName(_fromUtf8("lineCOM_compass"))
    self.pushButton_Run_compass =
        QtGui.QPushButton(rateDialog)
        self.pushButton_Run_compass.setGeometry(QtCore.QRect(470, 160, 75, 23))
    self.pushButton_Run_compass.setObjectName(_fromUtf8("pushButton_Run_compass"))

    self.pushButton_Stop_compass =
        QtGui.QPushButton(rateDialog)
    self.pushButton_Stop_compass.setGeometry(QtCore.QRect(470, 190, 75, 23))
    self.pushButton_Stop_compass.setObjectName(_fromUtf8("pushButton_Stop_compass"))

    self.pushButton_Close_compass =
        QtGui.QPushButton(rateDialog)
    self.pushButton_Close_compass.setGeometry(QtCore.QRect(470, 220, 75, 23))
    self.pushButton_Close_compass.setObjectName(_fromUtf8("pushButton_Close_compass"))

    self.pushButton_Init_compass =
        QtGui.QPushButton(rateDialog)
    self.pushButton_Init_compass.setGeometry(QtCore.QRect(640, 260, 75, 23))
    self.pushButton_Init_compass.setObjectName(_fromUtf8("pushButton_Init_compass"))

    self.kalmanDial = QtGui.QDial(rateDialog)
    self.kalmanDial.setGeometry(QtCore.QRect(990, 50, 161, 151))
    self.kalmanDial.setOrientation(QtCore.Qt.Horizontal)
    self.kalmanDial.setInvertedAppearance(False)
    self.kalmanDial.setInvertedControls(False)
    self.kalmanDial.setWrapping(True)
    self.kalmanDial.setNotchesVisible(True)
    self.kalmanDial.setObjectName(_fromUtf8("kalmanDial"))
self.label_7 = QtGui.QLabel(rateDialog)
self.label_7.setGeometry(QtCore.QRect(840, 10, 131, 31))
font = QtGui.QFont()
font.setPointSize(14)
self.label_7.setFont(font)
self.label_7.setObjectName(_fromUtf8("label_7"))
self.lcdkalman = QtGui.QLCDNumber(rateDialog)
self.lcdkalman.setGeometry(QtCore.QRect(840, 50, 121, 51))
self.lcdkalman.setObjectName(_fromUtf8("lcdkalman"))
self.label_8 = QtGui.QLabel(rateDialog)
self.label_8.setGeometry(QtCore.QRect(1070, 30, 46, 13))
font = QtGui.QFont()
font.setPointSize(11)
self.label_8.setFont(font)
self.label_8.setObjectName(_fromUtf8("label_8"))
self.pushButton_Reset = QtGui.QPushButton(rateDialog)
self.pushButton_Reset.setGeometry(QtCore.QRect(70, 170, 75, 23))
self.retranslateUi(rateDialog)
QtCore.QMetaObject.connectSlotsByName(rateDialog)

def retranslateUi(self, rateDialog):
rateDialog.setWindowTitle(QtGui.QApplication.translate("rateDialog", "Dialog", None, QtGui.QApplication.UnicodeUTF8))
self.label.setText(QtGui.QApplication.translate("rateDialog", "MLX90609", None, QtGui.QApplication.UnicodeUTF8))
self.label_2.setText(QtGui.QApplication.translate("rateDialog", "0", None, QtGui.QApplication.UnicodeUTF8))
self.pushButton_Run.setText(QtGui.QApplication.translate("rateDialog", "Run", None, QtGui.QApplication.UnicodeUTF8))
self.pushButton_Stop.setText(QtGui.QApplication.translate("rateDialog", "Stop", None, QtGui.QApplication.UnicodeUTF8))
self.pushButton_Close.setText(QtGui.QApplication.translate("rateDialog", "Close", None, QtGui.QApplication.UnicodeUTF8))
self.label_3.setText(QtGui.QApplication.translate("rateDialog", "COM Port Number", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Init.setText(QtGui.QApplication.translate("rateDialog", "Initialize", None, QtGui.QApplication.UnicodeUTF8))

self.label_4.setText(QtGui.QApplication.translate("rateDialog", "HMC6352", None, QtGui.QApplication.UnicodeUTF8))

self.label_5.setText(QtGui.QApplication.translate("rateDialog", "N", None, QtGui.QApplication.UnicodeUTF8))

self.label_6.setText(QtGui.QApplication.translate("rateDialog", "COM Port Number", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Run_compass.setText(QtGui.QApplication.translate("rateDialog", "Run", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Stop_compass.setText(QtGui.QApplication.translate("rateDialog", "Stop", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Close_compass.setText(QtGui.QApplication.translate("rateDialog", "Close", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Init_compass.setText(QtGui.QApplication.translate("rateDialog", "Initialize", None, QtGui.QApplication.UnicodeUTF8))

self.label_7.setText(QtGui.QApplication.translate("rateDialog", "Kalman Filter", None, QtGui.QApplication.UnicodeUTF8))

self.label_8.setText(QtGui.QApplication.translate("rateDialog", "N", None, QtGui.QApplication.UnicodeUTF8))

self.pushButton_Reset.setText(QtGui.QApplication.translate("rateDialog", "Reset", None, QtGui.QApplication.UnicodeUTF8))
The main code that was used to create the graphical user interface in Figure 21 is shown below

```python
import sys
import serial as communi
from PyQt4.QtCore import *
from PyQt4.QtGui import *
import Gyro_GUI as rate_gui

app = QApplication(sys.argv)
GUI = rate_gui.RateForm()
timer_gyro = QTimer()
timer_compass = QTimer()
count = 0
count_gyro = 0
Compass_initial = 0

def Process_Gyro_Data(data):
    global count_gyro
    global Compass_initial
    Angular_rate = 0
    Angular_rate = float(data.split(',',$)[1])
    GUI.DisplayRate_gyro(Angular_rate, Angular_rate)

def Init_Clicked():
    global arduino_port
    port_number = int(GUI.lineCOM.text())
    arduino_port = communi.Serial(port=port_number,
                                   baudrate=9600,
                                   bytesize=communi.EIGHTBITS,
                                   parity=communi.PARITY_NONE,
                                   stopbits=communi.STOPBITS_ONE,
                                   timeout=None,
                                   xonxoff=False,
                                   rtscts=False,
                                   dsrdtr=False,
                                   writeTimeout=False,
                                   interCharTimeout=None)

def RESET_Clicked():
    rate_gui.accumulated_value = 0
    rate_gui.accumulated_rotations = 0

def RUN_Clicked():
    timer_gyro.start(100)

def STOP_Clicked():
    timer_gyro.stop()
```
def CLOSE_Clicked():
    timer_gyro.stop()
    arduino_port.close()
    sys.exit()

###############################Kalman Code###############################

def Process_Kalman_Data(data):
    kalman_value = float(data.split(',')[3])  # was1
    kalman_value = round(kalman_value, 1)
    kalman_value = kalman_value % 360
    if (kalman_value > 180):
        kalman_heading = kalman_value - 360
    else:
        kalman_heading = kalman_value
    GUI.DisplayRate_kalman(kalman_value, kalman_heading)

###############################Compass Code###############################

def Process_Compass_Data():
    heading = 0
    global Compass_initial
    global count
    data = arduino_port.readline()
    print(data)
    heading_value = float(data.split(',')[0])
    if (heading_value > 180):
        heading = heading_value - 360
    else:
        heading = heading_value
    GUI.DisplayRate_compass(heading_value, heading)
    if (count == 0):
        Compass_initial = heading_value
        count = count + 1
    Process_Gyro_Data(data)
    Process_Kalman_Data(data)

    timer_compass.timeout.connect(Process_Compass_Data)

    def INIT_Compass_Clicked():


global arduino_port

port_number = int(GUI.lineCOM_compass.text())
arduino_port = communi.Serial(port=port_number,
baudrate=115200, bytesize=communi.EIGHTBITS,
parity=communi.PARITY_NONE, stopbits=communi.STOPBITS_ONE,
timeout=None, xonxoff=False, rtscts=False, dsrdtr=False,
writeTimeout=False, interCharTimeout=None)
arduino_port.close()

def RUN_Compass_Clicked():
    global arduino_port

    arduino_port.open()
timer_compass.start(180)
    #timer_gyro.start(200)

def STOP_Compass_Clicked():
    global arduino_port

    timer_compass.stop()
    arduino_port.close()

def CLOSE_Compass_Clicked():
    timer_compass.stop()
    arduino_port.close()
    sys.exit()

GUI.pushButton_Init.clicked.connect(Init_Clicked)
GUI.pushButton_Reset.clicked.connect(RESET_Clicked)
GUI.pushButton_Run.clicked.connect(RUN_Clicked)
GUI.pushButton_Stop.clicked.connect(STOP_Clicked)
GUI.pushButton_Close.clicked.connect(CLOSE_Clicked)
GUI.pushButton_Init_compass.clicked.connect(INIT_Compass_Clicked)
GUI.pushButton_Run_compass.clicked.connect(RUN_Compass_Clicked)
GUI.pushButton_Stop_compass.clicked.connect(STOP_Compass_Clicked)
GUI.pushButton_Close_compass.clicked.connect(CLOSE_Compass_Clicked)
GUI.show()
app.exec_()
Appendix C – Quotation for LT3824 5Volt regulator circuit

Pricing is valid for this Web ID until 3/1/2012 if you place your order online.

If you need to find a part number, use the search box in the upper right of the page or click the icon here. If you know exactly what you want, fill in the quantity, part number, and optional customer reference number below and select Add to Order. Quantities are not reserved until the order has been submitted. You will be reminded of backorder quantities at submit time. Available quantities may have changed due to real-time inventory. To check current availability, click the “Update Stock Status” button below.

<table>
<thead>
<tr>
<th>Index</th>
<th>Quantity</th>
<th>Part Number</th>
<th>Description</th>
<th>Customer Reference</th>
<th>Backorder Quantity</th>
<th>Unit Price</th>
<th>Extended Price</th>
</tr>
</thead>
</table>