

Self-Balancing Robot

Third Year Individual Project – Final Report
April 2017

Abdul Gafar

9097951

Supervisor: Dr Joaquin Carrasco Gomez

Acknowledgements

First, I would like to thank my parents and my brother for their unconditional support over the last three years.

I would like to express my sincere gratitude to Dr Carrasco for accepting to be my Supervisor for this 3rd year project. I am grateful for the help and advice provided throughout the year, for without it I would have not progressed as far as I did.

Lastly, I would to thank my friends Omar El-Shabassy, Andrei Velcescu and Canberk Gurel for their support and help thought the whole project.

Abstract

Self-balancing robots are a topic of curiosity amongst students, roboticists, and hobbyists around the world. The fascinating aspect is the fact that it is a naturally unstable system. This project presents an attempt on developing an autonomous self-balancing robot. A key element in maintaining the robot in the upright position is estimation of the tilt angle. For this, the Kalman Filter has been implemented and tested to fuse data from a gyroscope and an accelerometer. In addition, the methodology in which the hardware was chosen and put together has been justified. Then the software development and challenges in the implementation of the Kalman Filter have also been explained. Lastly the control of the robot has been explored, with characterisation of the robot in an attempt to implement an LQR controller. Although several results were presented, the goal of making a self-balancing robot was not achieved.

Contents

1. Introduction, Motivation and Aims	1
1.1. Introduction	1
1.2. Motivation.....	1
1.3. Aims and Objectives	2
2. Background Information and Literature Review	3
2.1. Introduction	3
2.2. Inverted Pendulum Systems	3
2.2.1. Simple Pendulum	4
2.2.2. Non-Minimum Phase Zeros and Transfer Function Analysis.....	4
2.3. Controllers.....	5
2.4. Tilt Angle Estimation	6
2.5. Summary.....	7
3. Sensor Fusion.....	8
3.1. Introduction	8
3.2. Gyroscope Fundamentals	9
3.3. Accelerometer Basic Principles.....	10
3.4. Kalman Filter.....	11
3.4.1. Background	11
3.4.2. Kalman Filter Algorithm	12
3.4.3. Model derivation	13
3.4.4. Overall Diagram.....	15
3.4.5. Initialisation.....	15
3.4.6. Tuning Parameters	17
3.5. Complementary Filter.....	20
3.6. Analysis of Performance	21
4. Hardware	23
4.1. Introduction	23
4.2. Motors and Wheels	23
4.3. Microcontroller	24

4.4.	Inertial Measurement Unit (IMU)	25
4.5.	Motor Driver Board.....	26
4.6.	Power Source	26
4.7.	Overall Design	26
4.8.	Final Assembly.....	27
5.	Software Implementation	28
5.1.	Introduction	28
5.2.	Constant Loop Time.....	28
5.3.	Reading from IMU.....	29
5.4.	KF Implementation	30
5.5.	Displacement and Angular Velocity from Encoders	30
5.6.	Overall Diagram	31
6.	Control	33
6.1.	Introduction	33
6.2.	State-Space Model	33
6.3.	Estimation of the Robot's Parameters.....	35
6.4.	Analysis of the System.....	36
6.5.	LQR in MATLAB	36
6.6.	Physical Performance	38
7.	Conclusions	39
7.1.	Project Achievements	39
7.2.	Project Limitations and Final Remarks.....	39
8.	References.....	41
9.	Appendices	45
9.1.	Appendix 1 – MATLAB code.....	45
9.2.	Appendix 2 – Arduino PID code	46
9.3.	Appendix 3 – Arduino Motor Characterisation Code	53
9.4.	Appendix 4 – Arduino LQR Code.....	56
9.5	Appendix 5 - Progress Report.....	64

1. Introduction, Motivation and Aims

1.1. Introduction

Self-balancing robots have been a topic of interest of many researchers, students and hobbyists worldwide. In essence, it is an inverted pendulum on wheels, a derivative of the inverted pendulum on a cart. Unlike traditional robots, which are in a constant state of equilibrium, the robot is a naturally unstable system [1]. Its design is more complex, as it needs to be actively controlled to maintain its upright position, however, it benefits from being able to turn on the spot.

The primary practical application of a self-balancing robot is human transportation, which was popularised by the release of the Segway PT (Personal Transporter) [2]. It is used in many industries such as inside factory floors or for tourism in the park. It is more attractive compared to four or three wheeled vehicles as they can take sharp turns and navigate in tighter spaces [3].

1.2. Motivation

The primary incentive of the project is to develop general understanding of control theory. For the last few decades, “the inverted pendulum has been the most popular benchmark, among others, for teaching and research in control theory and robotics [4].” Hence, developing a self-balancing robot is the ideal platform to put into practice what has been covered in Control Systems lectures. It would also be interesting to see the differences between the behaviour in practice compared to simulations. Furthermore, the material and methods learnt have a wide array of applications; for example, inverted pendulums have been used to model human locomotion, which then was used to develop bipedal robots [5].

In addition to control theory, learning about Kalman Filters (KF) was also a motivation to develop the self-balancing robot. Knowing the tilt angle is necessary in any balancing robot to apply the appropriate control action. However, different sensing devices have their compromises; the KF is used to fuse data from two sensors, such that a better estimate of the tilt angle can be obtained. Kalman filters specifically and not just the Complementary Filter, because it is considered to be “one of the most important data fusion algorithms in use today [6]” and it was famously used in the first manned mission to the moon [7].

Besides learning about the theoretical aspects, the project also incorporated a practical side. This includes but is not limited to, using SolidWorks to see how everything was going to fit together, using stripboards for small-simple circuits, using lab equipment for testing and programming in C. These are a wide array of important skills applicable to many tasks, in future projects, as an engineer.

1.3. Aims and Objectives

This project aims to design, construct and program a self-balancing robot with a self-developed and implemented Kalman filter. To achieve the aims of the project, following objectives have been set:

- Perform Literature Review on Kalman Filters and implement it in MATLAB
- Develop the mathematical model for sensor fusion
- Implement and tune the Kalman Filter in a microcontroller
- Design a testing rig to the Kalman Filter
- Tune the KF to have the best performance
- Design and assemble the chassis of the robot
- Develop the software to read from the sensors and to control the actuators
- Implement a PID controller to enable the robot to stay upright

2. Background Information and Literature Review

2.1. Introduction

This chapter aims to provide an overview of the literature sources used throughout the development of the robot. First, the fundamentals of inverted pendulum systems are described: determination of the equilibrium point and what makes the system interesting to control engineers. Afterwards, a literature review on the most common control theory applied to self-balancing robots is performed and the last sub-section summarizes the main sensor fusion techniques.

2.2. Inverted Pendulum Systems

The inverted pendulum is a classical problem in control systems, and to explore the unstable dynamics, different platforms have been developed. These platforms are similar in many ways, leading to many of the behaviours being comparable. The most common types are the self-balancing robot, Inverted Pendulum on a cart and an inverted pendulum on a linear track, shown in the figure below:

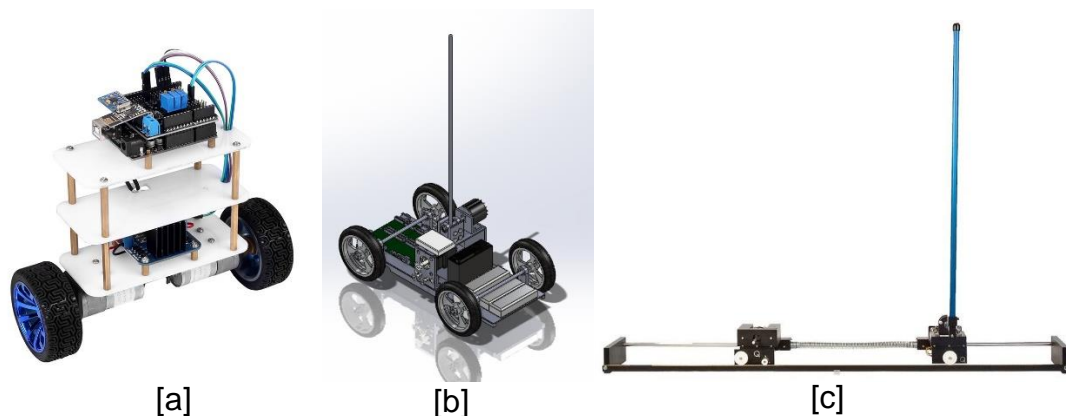


Figure 1: [a] Self-Balancing Robot [8], [b] Inverted Pendulum on a Cart [9], [c] Inverted Pendulum on a Linear Track [10]

2.2.1. Simple Pendulum

To better understand these systems, analysis of the dynamics of a simple pendulum is crucial.

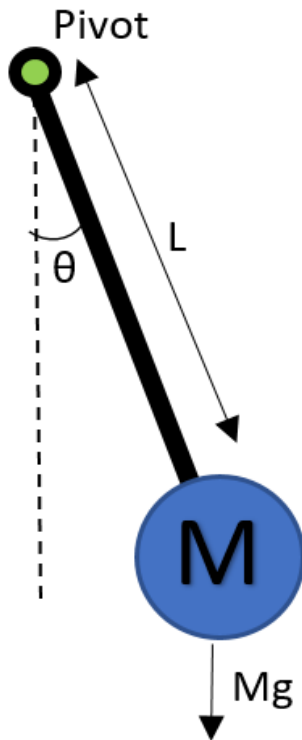


Figure 2: Simple Pendulum

Assuming the system on the left, where a mass, M is connected by a massless rod of length L to a frictionless pivot. The angular velocity, ω and the rate of change of angular velocity, $\frac{d\omega}{dt}$ are given by the following formulae [11]:

$$\frac{d\theta}{dt} = \omega \quad (1)$$

$$\frac{d\omega}{dt} = -\frac{MgL}{J} \sin \theta \quad (2)$$

Where J is the moment of inertia, and g is the gravitational force

Intuitively, the system will remain at rest when $\omega = 0$ and $\frac{d\omega}{dt} = 0$. When equation 1 and 2 are solved to fit the criteria, the equilibrium points are

found to be $\dot{\theta} = 0$, and $\theta_1 = 0$ or $\theta_2 = 180^\circ$. θ_1 is stable, but not applicable to self-balancing robots, θ_2 is the target position for inverted-pendulum systems. However, at this position, the system is unstable. Any external disturbance will cause the pendulum to move indefinitely away from that specific equilibrium point, hence the need for it to be actively balanced.

2.2.2. Non-Minimum Phase Zeros and Transfer Function Analysis

Perhaps the most fascinating aspect of inverted pendulum systems is the notion of non-minimum phase zeros. Following [12], a summary of the consequences of the non-minimum phase zeros is given. As explained in the previous section, inverted pendulums are unstable, attempting to stabilise an unstable plant using feedback will give rise to non-minimum phase zeros. The effect of these zeros can be observed by a simple experiment of trying to balance a long stick on the palm of one's hand. The stick is analogous to the inverted pendulum and the hand to the cart. When attempting to move the hand to right, initially one tends to move to the left briefly then move to right. The zero in the system's transfer function has caused an initial

undershoot when a 'step response' is applied. The conclusive effect of non-minimum phase zeros will vary depending on the controller used, it may cause initial undershooting, direction reversals and/or overshooting.

Given the transfer function $G(s)$,

$$G(s) = \frac{N(s)}{D(s)} \quad (3)$$

a zero is a the root in the numerator, $N(s)$, of $G(s)$. A non-minimum phase zero is a positive zero that is in the right half of the pole-zero plot. To exhibit the behaviour, in which direction reversals occurs, has to be an odd number of zeros in the transfer function.

From the transfer function, further conclusions can be made. If there are positive poles, the plant is unstable and based on parity interlacing principle, if the plant has odd number of positive real poles to the right of the non-minimum poles the plant cannot be controlled by a stable controller. This is actually present in a linearized transfer function of an inverted pendulum on a cart. Further discussion in context of the self-balancing robot designed is given in Chapter 6.

The overall effect of the non-minimum phase zeros is constraints in closed loop performance. They restrict the bandwidth and it causes a limited gain margin (suggesting limited robustness). The non-minimum phase zeros cannot be cancelled in practice as a small difference in the zero and poles will lead to instability in the plant.

2.3. Controllers

To maintain the robot upright, the most commonly used controllers are Proportional-Integral-Derivative (PID) and the Linear Quadratic Regulator. Other theses have also explored the use of Linear-Gaussian Control (LQG), Fuzzy Logic and Pole-Placement; however, in some cases they were never implemented in a robot and were only experimented in simulations [13]. In theses where the robot displacement is also controlled, either LQG is used or combination of controllers. For example LQR to maintain the robot upright and PID for controlling displacement, or a cascaded PID controller [14] [13] [15] [16] [17] [2] [1]. PID and LQR are the explored controllers in this thesis, thus further details will only be provided for them.

PID is perhaps the most used controller, as stated by VanDoren “More than 60 years after the introduction of proportional-integral-derivative controllers, they remain the workhorse of industrial process control [18].” The algorithm is described by the following equation:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{d}{dt} e(t) \quad (4)$$

Where, $u(t)$ is the output of the controller, $e(t)$ is the error and K_p , K_i and K_d are the tuning parameters. It is relatively easy to implement and does not require a model of the system. Tuning of the parameters can be done with trial and error. Using this simple method, robots have been controlled to remain upright [13] [15] [16] [2] [1] [17].

LQR is a form of optimal control that aims to minimize the performance index whilst taking into account the control effort, as often, higher input effort would imply higher energy consumption [19]. LQR control requires derivation of the state-space model of the system [14], thus it is more challenging to implement. A great advantage of LQR is that unlike PID it can be applied to Multiple Input, Multiple Output (MIMO) systems. When applied to the self-balancing robot, it was found that LQR has a better performance [20].

2.4. Tilt Angle Estimation

In order to maintain the robot upright, knowing the tilt angle is imperative. There are a wide array of sensors that can be used, such as inclinometers, light sensors, accelerometer or gyroscopes. However, each these sensors have their shortcomings, the inclinometer takes a long time to converge to the angle it is currently at, light sensors are highly susceptible to background noise (ambient light and the reflective index of the surface it is operating in), gyroscopes have a bias and accelerometers are relatively noisy.

Most often, a combination of a gyroscope and an accelerometer are used. To combine the measurements, the most commonly used techniques are the Kalman Filter and the Complementary Filter. In more modern Inertial Measurement Units (IMU) built in algorithms can be found to fuse the data. The next paragraphs, will provide an overview of each of the mentioned techniques, and the findings from the literature review conducted. There is a greater emphasis on the Kalman Filter as it is one of the core aspects to be explored in the project.

Rudolf Kalman introduced the Kalman Filter, also known as Linear Quadratic Estimation, in 1960 [3]. “If implemented and tuned correctly, the Kalman Filter best possible (optimal) estimator for a large class of problems [21].” To implement the Kalman Filter, a state space model of the system is required. One of the great advantages, in addition to being a good estimator, is the fact that it is a recursive method. This means that large amount of data does not need to be stored and can run in real time, allowing it to be implemented in devices that have lower memory sizes [3]. The challenge for someone without previous experience in using KFs, is confusion caused by varying notations in textbooks [3].

The complementary filter is simply composed of a high pass filter for the integrated data from the gyroscope and a low pass filter for the angle calculated from the accelerometer [22]. It is the most commonly used method as it is relatively easy to implement. Bonafilia et al, have found that it performs better than the Kalman filter [14].

In some modern IMUs, such as InvenSense's MPU 6050 or Bosch's BNO055, there is built-in sensor fusion algorithm's that will directly output the absolute orientation. This removes the load from the main microcontroller and the algorithm is expected to have the best performance as it has been developed by the manufacturer [23] [24].

2.5. Summary

To begin with, the fundamentals of the system were explored and it was established that it the robot is at equilibrium when upright or at least the centre of mass is above the pivot point. Then, it was revealed that the robot has non-minimum phase response in closed loop. Finally, an overview of the most commonly used sensor fusion techniques and controllers. The findings are summarized in the tables 1 and 2:

Controller	Advantages	Disadvantages
PID	<ul style="list-style-type: none"> - Easy to implement -Does not require the state space model 	<ul style="list-style-type: none"> - Applicable to Single Input, Single Output Systems (SISO) - Does not perform as well as LQR
LQR	<ul style="list-style-type: none"> - Applicable to MIMO systems - Performs better than PID 	<ul style="list-style-type: none"> -More challenging to Implement -Requires derivation of the State-Space Model

Table 1: Comparison of Controllers

Angle Estimation Techniques	Advantages	Disadvantages
Kalman Filter	<ul style="list-style-type: none"> - Regarded as one of the best estimators - It is recursive method 	<ul style="list-style-type: none"> - Requires a state-space model of the system - Difficult to understand due to standard notation -Has higher computational requirements
Complementary Filter	<ul style="list-style-type: none"> - Easy to implement 	<ul style="list-style-type: none"> - Can be susceptible to noise
Built-In Algorithm	<ul style="list-style-type: none"> -Removes the load from the main microcontroller 	<ul style="list-style-type: none"> - Not available in every IMU

Table 2: Comparison of Angle Estimation Techniques

3. Sensor Fusion

3.1. Introduction

A common agreement in literature is that using either a gyroscope or an accelerometer on their own to obtain the tilt angle is not very reliable. This primarily arises from the fact that both of these devices have a bias in the measurements, are affected by white noise and the bias is affected by temperature. Attempting to account for all the errors, could be a dissertation in itself [25] . In this project, as only the tilt angle is measured and not relative displacement, it is assumed that the gyroscope is mainly affected by a bias and the accelerometer by white noise.

This chapter begins by presenting the fundamentals of gyroscopes and accelerometers to highlight the need for sensor fusion. Followed by, the Kalman Filter and its implementation, the complimentary filter and finally a comparison of each of the filters. The Kalman Filter is covered in greater detail as it is one of the primary focuses of this thesis. The complementary filter has solely been explored to compare whether the additional computational requirement is a significant improvement when used to estimate the tilt angle.

3.2. Gyroscope Fundamentals

The gyroscope measures angular velocity, $\dot{\theta}$, in radians per second or degrees per second. Intuitively, by integrating the angular velocity the tilt angle can be calculated. Since the gyroscope readings are taken at discrete time intervals, dt , numerical integration is performed using the Euler method. This is shown in the equation below:

$$gyro_{angle}_t = gyro_{angle}_{t-1} + angular_{velocity} * dt \quad (5)$$

The equation above assumes the sensor initial position is 0 degrees. If the starting position of the sensor is non-zero, the angle has to be initialized to the value, possibly from another sensor i.e. inclinometer or accelerometer.

Using the approach mentioned the following graph was obtained:

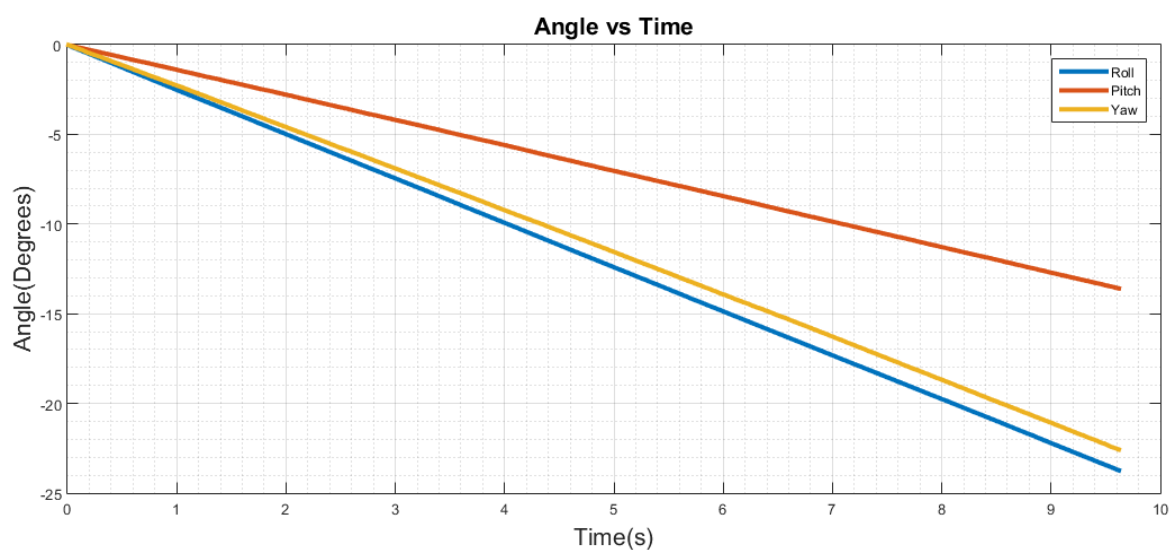


Figure 3: Graph of Angle (calculated using equation 1) against Time

A key element to observe was that the measurements were taken while the IMU was at rest; this highlights the effect of the bias in the measurements. Due to the integration, this systematic error is summed in every loop, thus the angle appears to be increasing even though the gyroscope was not moving. Furthermore, the size of the bias also increases over time [26]. In cases where the object can rotate in 3D, the yaw angle needs to be coupled with the roll and pitch to reduce the chances of Gimbal Lock [27]. The robot is assumed to have only one axis of rotation thus coupling does not need to be considered

3.3. Accelerometer Basic Principles

The accelerometer measures the acceleration relative to free fall. The acceleration is often measured in gs, which is based on earth's gravitational pull (9.81m/s). To determine the orientation of the accelerometer, it is assumed that the only force acting on the object is earth's gravitation pull. Gravity always acts 'down', thus when the object is tilted, the force is divided into components in the x, y, and z directions of the object. Since the axes are orthogonal to one another, Pythagoras theorem can be used to show the relationship between the forces as shown in the diagram below:

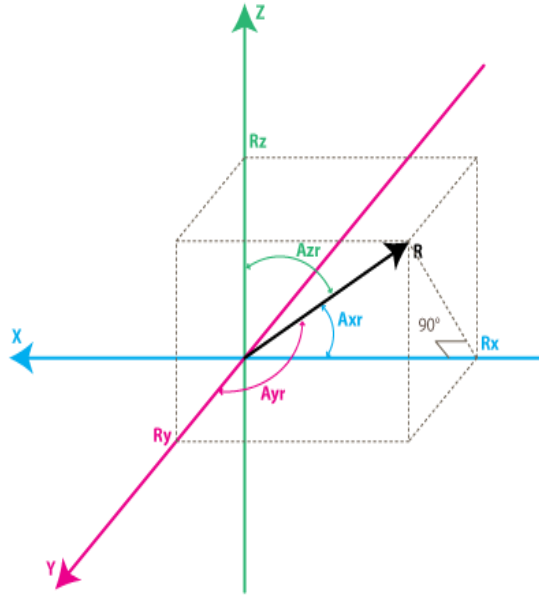


Figure 4: Diagram that shows the components of gravity in each direction [26]

From the diagram, the following equation can be obtained:

$$A_{xr} = \theta_x = \cos^{-1} \left(\frac{x}{\sqrt{x^2 + y^2 + z^2}} \right) = \cos^{-1} \left(\frac{x}{R} \right) \quad (6)$$

Alternatively, the following equation is more useful as it calculates the angle relative to the z-axis (Roll) [28]:

$$Roll = \phi_{xyz} = \tan^{-1} \left(\frac{y}{z} \right) \quad (7)$$

With a similar approach, Pitch can also be calculated. However, Yaw cannot be determined accurately, especially when the force in the z direction = 1g (assuming the accelerometer can only rotate and not translate in any direction). In this case changing the yaw, will have no impact on the components of x and y, making yaw constant. This however does not affect the implementation in the robot as only the tilt angle is required [26].

Using equation 6, the following graph was obtained:

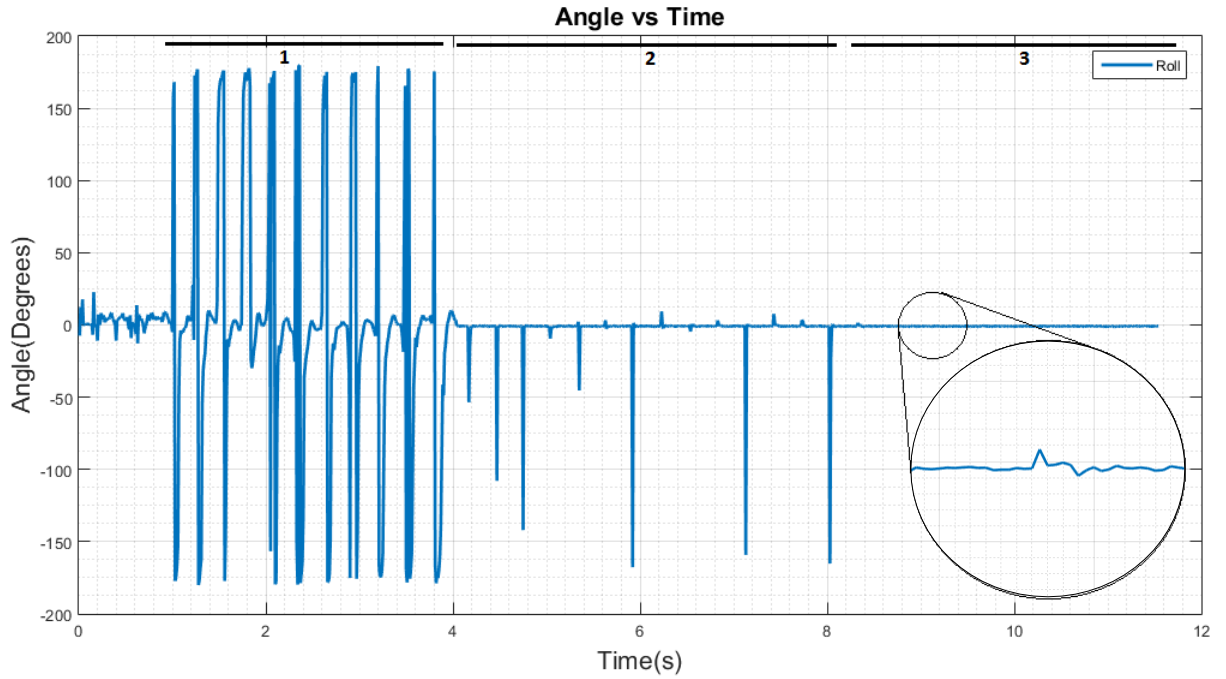


Figure 5: Graph of Angle (Accelerometer) against Time

The graph above is divided into 3 sections. In all cases, the IMU orientation was kept constant (as well as humanly possible). In section 1, the IMU was moved up and down. In section 2, the table was tapped lightly and in section 3, the accelerometer was left without external disturbances. The diagram shows the problems with using accelerometer for measuring angles, even in section 3, when zoomed in it can be see that the line is still not perfectly smooth, emphasising the noisy nature of the accelerometer. The root of these issues is the assumption that gravity is the only force acting upon the object [29].

3.4. Kalman Filter

3.4.1. Background

As microcontrollers work in discrete time, the discrete time Kalman Filter will be used to estimate the tilt angle. To implement the algorithm, the process has to be described by the linear stochastic difference equation [21]

$$x_k = Ax_{k-1} + Bu_k + w_k \quad (8)$$

where, x_k is the state vector containing the variables to be estimated. A is the state transition matrix that is applied to x_{k-1} . u_k is the control (input) vector and B (control

input matrix) maps the inputs to the state vector. Finally, w_k is vector that contains the process noise for each of the variables in x_k . The noise is assumed to be normally distributed with the a mean value of zero and the covariance is given by Q

$$p(w) \sim N(0, Q) \quad (9)$$

The measurements are modelled by

$$z_k = Hx_k + v_k \quad (10)$$

Where, z_k is contains the measured values of x_k . H transforms the state vector into measurements. v_k is the measurement noise also Gaussian distributed but with a variance of R

$$p(v) \sim N(0, R) \quad (11)$$

w_k and v_k are white noises and independent of each other [30].

3.4.2. Kalman Filter Algorithm

The KF implements a form of feedback: first, a process state estimate is made using the time update equations and then using the measurement update equation a form of measurement estimation is obtained [21]. The equations for the time update are presented below:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k \quad (12)$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k \quad (13)$$

Where \hat{x} is the state estimate and P is the process covariance matrix.

The measurement update is described by the following equations:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \quad (14)$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H\hat{x}_{k|k-1}) \quad (15)$$

$$P_{k|k} = P_{k|k-1} - K_k H_k P_{k|k-1} \quad (16)$$

Where K is the Kalman Gain Matrix.

The time update equations aim to predict the current state and process covariance given the information of the previous steps, this is known as the priori state. The measurement equations form part of the feedback where the measurements of the current state are incorporated into the priori state forming the posteriori estimate, $\hat{x}_{k|k}$ and $P_{k|k}$. This notion of priori and posteriori is shown by the subscript in the equations, where $\mathbf{a} | \mathbf{b}$ would mean an estimate of \mathbf{a} based on \mathbf{b} and all previous states before \mathbf{b} .

In equation (15) the part in parentheses, $z_k - H\hat{x}_{k|k-1}$, calculates the difference between the predicted value and the measured value, this is known as the innovation or residual represented by as y_k and in equation (14) the section $H_k P_{k|k-1} H_k^T + R_k$ is known as the innovation covariance, commonly denoted as S_k . Using the S_k the Kalman Gain, K_k , is calculated which is then forms part of (15) to compute the posteriori state estimate $\hat{x}_{k|k}$. The fundamental part of the KF is the calculation of the K_k , from a series of substitutions and manipulations, it aims to minimize the posteriori error covariance (16) [21]. The overall effect can be visualised in the diagram below:

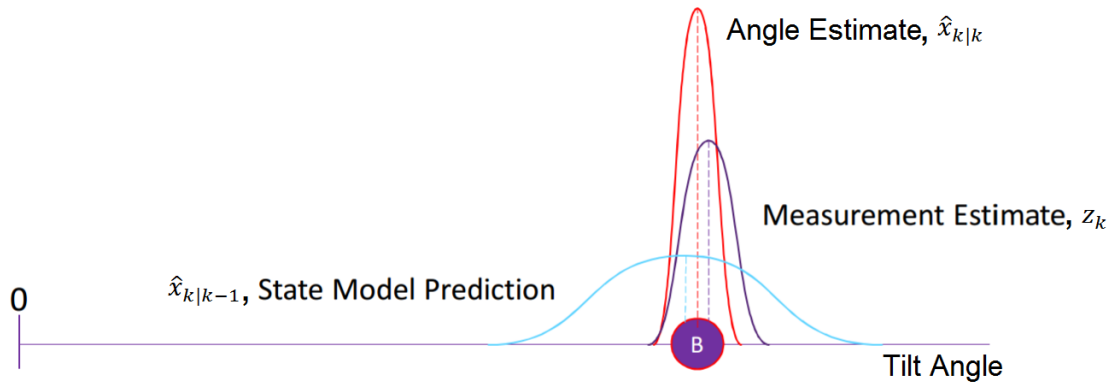


Figure 6: Diagram showing the effect of the KF estimation, adapted from [31]

3.4.3. Model derivation

As shown by Gorniki, the model can be derived from continuous time, using the assumptions that the measurement from the gyroscope outputs, U , is composed of the angular velocity with a constant bias [16] [32]

$$U = \dot{\theta} + \theta_{bias} \quad (17)$$

$$\dot{\theta}_{bias} = 0 \quad (18)$$

Equation (17) is re-arranged to make $\dot{\theta}$ the subject:

$$\dot{\theta} = -\theta_{bias} + U \quad (19)$$

The model is discretised in step intervals of k using the approximation (20) [31]

$$\frac{dx}{dt} = \frac{\Delta x}{\Delta t} = \frac{x_k - x_{k-1}}{t_k - t_{k-1}} \quad (20)$$

$$\frac{\theta_k - \theta_{k-1}}{dt} = -\theta_{bias_k} + U_k \quad (21)$$

$$\frac{\theta_{bias_k} - \theta_{bias_{k-1}}}{dt} = 0 \quad (22)$$

Re-arranging to make θ_k and θ_{bias_k} the subjects of the equations:

$$\theta_k = \theta_{k-1} - \theta_{bias_k} \cdot dt + U_k \cdot dt \quad (23)$$

$$\theta_{bias_k} = \theta_{bias_{k-1}} \quad (24)$$

Equation (24) is similar to the equation (4) except here the bias is also considered.

Equation (23) and (24) can now be put in the state space format:

$$\begin{bmatrix} \theta \\ \theta_{bias} \end{bmatrix}_k = \begin{bmatrix} 1 & -dt \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \theta_{bias} \end{bmatrix}_{k-1} + \begin{bmatrix} dt \\ 0 \end{bmatrix} U_k \quad (25)$$

The accelerometer then comes into the algorithm in the measurement update. Using the equation (d) the tilt angle can be calculated but not the bias, thus the H matrix is given by:

$$H = [1 \quad 0] \quad (26)$$

3.4.4. Overall Diagram

The diagram below best describes the algorithm:

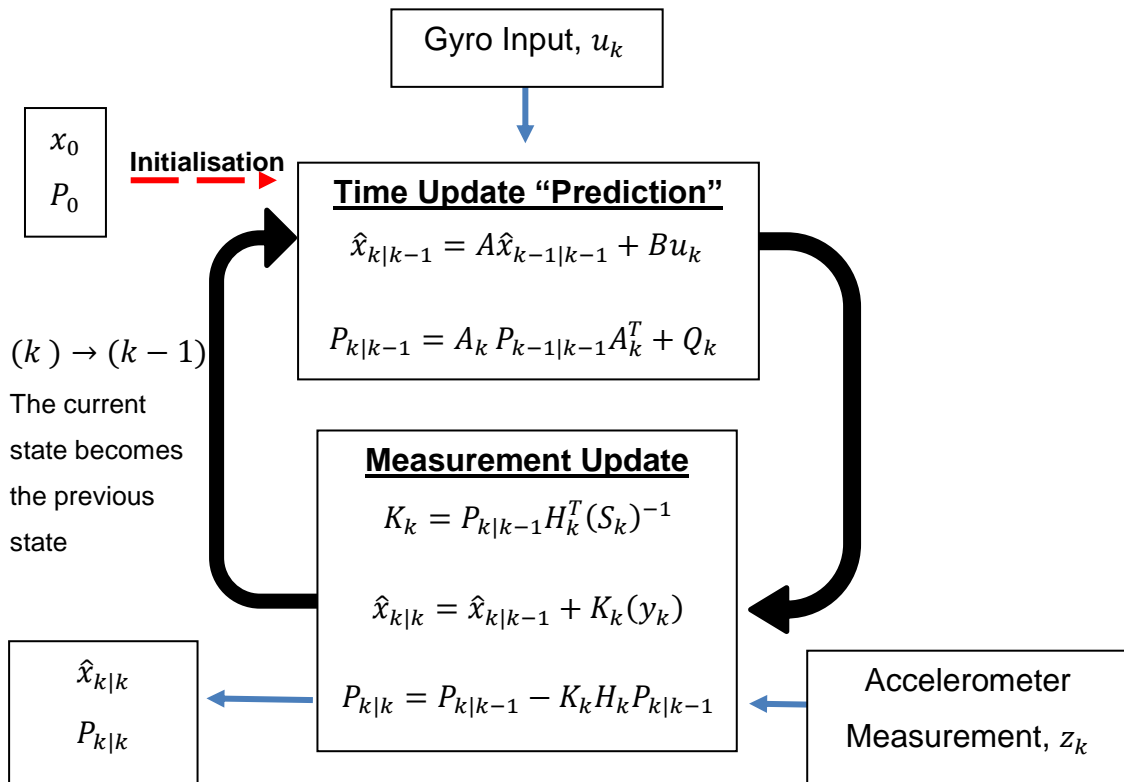


Figure 7: Ongoing Discrete Kalman Filter Cycle [34]

3.4.5. Initialisation

The model derived in section 4.4.3 calculates the tilt angle relative to an initial position; the initial position is measured from the accelerometer or another alternative is to start the algorithm whilst the robot is vertical and set the initial tilt angle to zero. The initialising bias can be calculated by taking a few readings while the gyroscope is still and then take the average or just setting it to zero.

The process covariance matrix also needs to be initialised. This will depend on the initialisation of the state vectors. If the initial state vector is initialised with a well estimated value then P_0 can be set to a diagonal matrix with relatively small values,

$$P_0 = \begin{bmatrix} \text{Small} & 0 \\ 0 & \text{Small} \end{bmatrix} \quad (27)$$

on the other hand if the initial state vector is initialised to badly estimated values then P_0 is set to a diagonal matrix with relatively high values [16],

$$P_0 = \begin{bmatrix} \text{Large} & 0 \\ 0 & \text{Large} \end{bmatrix} \quad (28)$$

In more precise terms, “the values in the covariance matrix must be defined such that the difference between the initial state and the initial state estimate fall in the range

that is allowable according to the covariance matrix.” Thus the initialisation of the state vector does not have to be very accurate, after some iterations the KF will converge to the actual value, given that P_0 is sufficiently large.

Overall, there are two cases and the effects of each can be seen in the diagrams below:

- Case 1: High P_0 and the state vector has been initialised to zero although the sensor was tilted

$$P_0 = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}$$

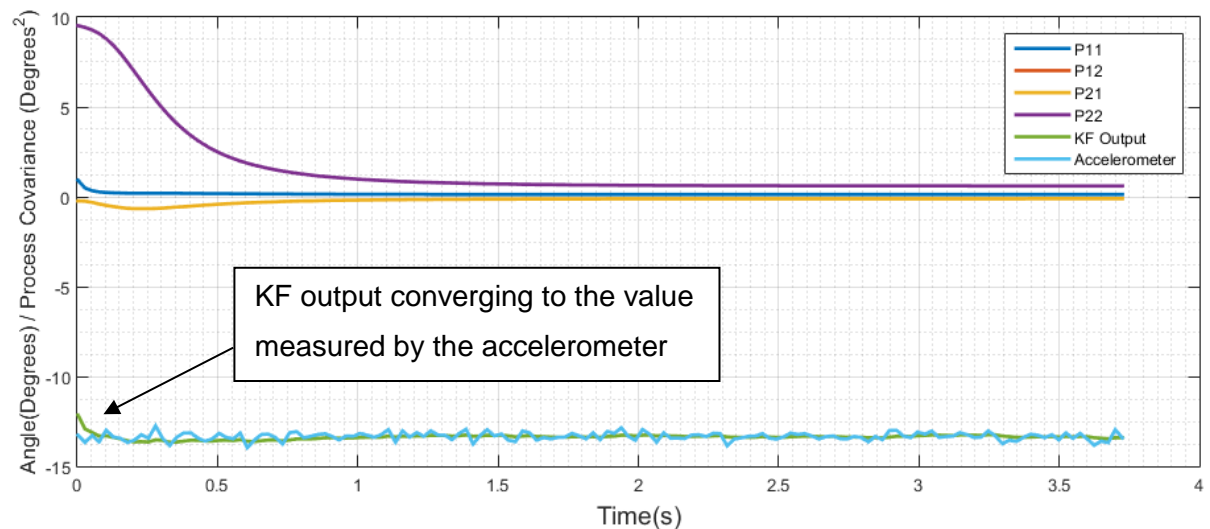


Figure 8: Graph showing the effect of Case 1

- Case 2: Low P_0 and the state vector has been initialised using the angle from the accelerometer and the bias has been estimated by taking the average of a few readings.

$$P_0 = \begin{bmatrix} 0.05 & 0 \\ 0 & 0.05 \end{bmatrix}$$

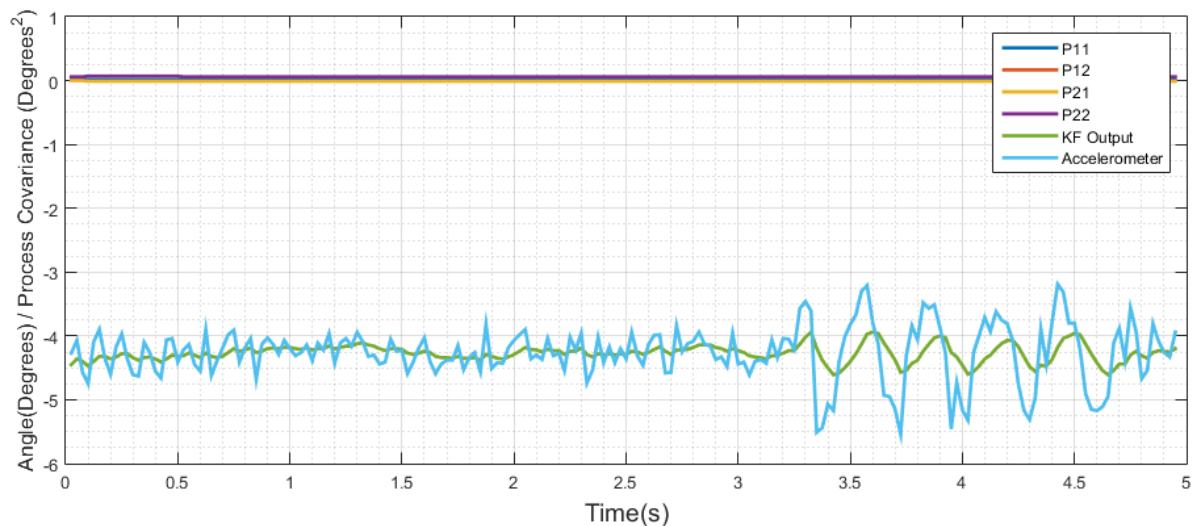


Figure 9: Graph showing the effect of Case 2

In case one, as can be seen from the graph the KF output takes about 0.2 seconds to converge to the value of the accelerometer reading. The elements in P take about 2 seconds stabilise. In case two, the KF output starts with the accelerometer, change in elements of P are not even visible through the serial monitor, this is due to limit in decimal places in the serial communication. As the output stabilises relatively quickly, the benefits of both scenarios are exploited, the tilt angle is initialised from the accelerometer, the bias by taking the average over a certain period and P_0 with a relatively high value.

3.4.6. Tuning Parameters

The final requirement to apply the KF and have a good performance is appropriate tuning. The tuning parameters are the measurement noise covariance, R_k , and the process noise covariance, Q_k . R_k can be measured as this is from the measurement device. Q_k on the other hand is difficult to measure as the process being estimated cannot be directly observed [3]. R_k was estimated by recording values whilst the accelerometer was stationary, then using the variance formula below where N is the number of samples, A the random variable and μ is the mean [33]:

$$\text{Covariance}(A, A) = \text{Variance}(A) = \frac{1}{N-1} \sum_{i=1}^N |A_i - \mu|^2 \quad (29)$$

$$\mu = \frac{1}{N} \sum_{i=1}^N A_i \quad (30)$$

With a sample of 900 measurements R_k was found to be 0.0527 degrees squared. Although the value of R_k was calculated, “superior filter performance (statically speaking) can be obtained by tuning the filter parameters. [21]” To do so, understanding of the effects of each of the parameters is essential.

R_k describes the precision of the measurements, it has an effect on the Kalman Gain (14) and the posteriori estimate (15). If R_k is large then the Kalman Gain will be smaller, hence there less ‘trust’ in the innovation and vice versa. The practical effect of changing R_k is shown in Figures 10 and 11 where the sensor was moved in a way that square waves were generated.

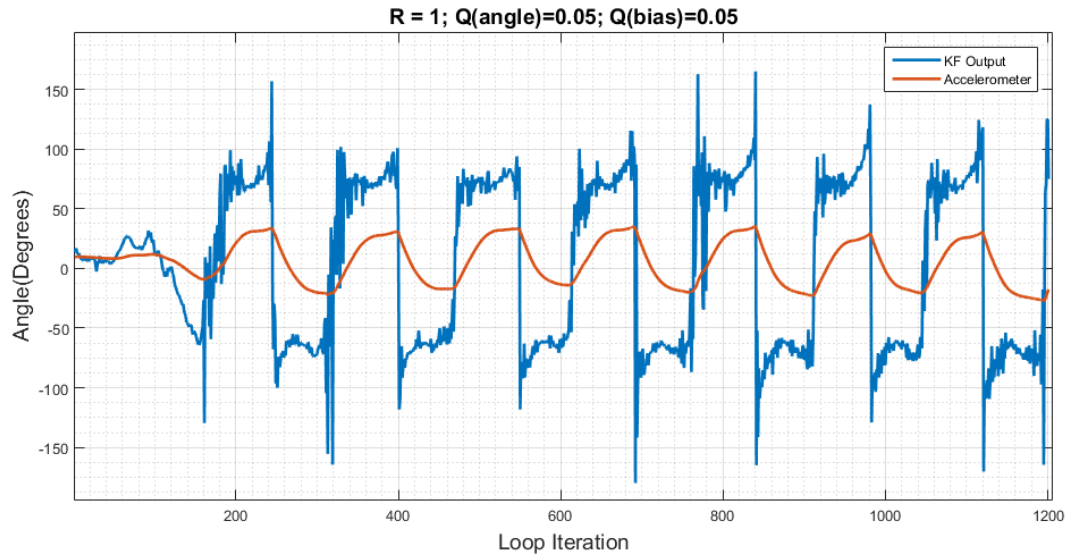


Figure 10: Graph Showing the effect of high R_k

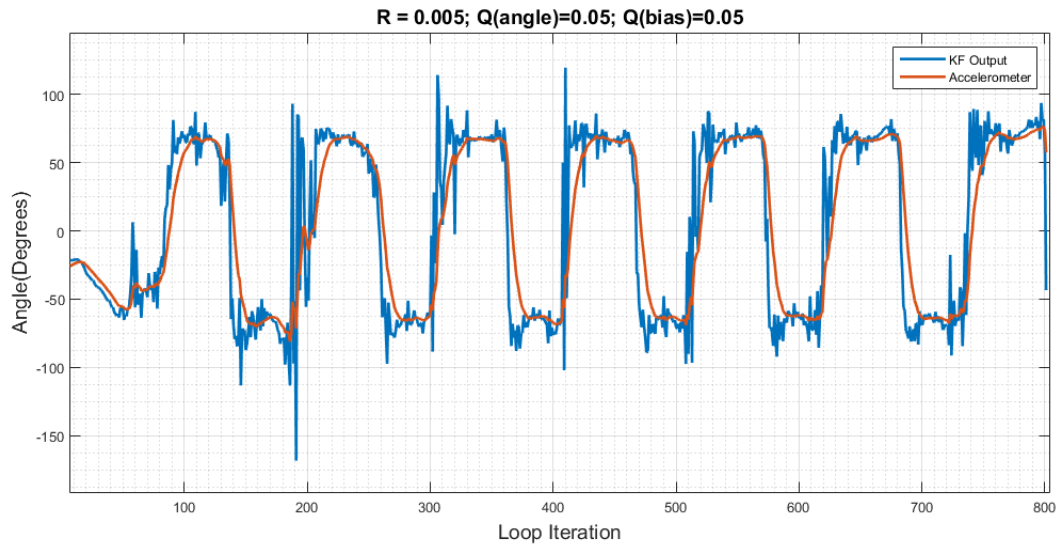


Figure 11: Graph Showing the effect of low R_k

From Figure 10, it can be observed that setting R_k to a value that is too large leads to the output of the KF taking too long to converge causing the output to have a smaller amplitude than the actual movements and the output not being square waves, instead being composed of smooth curves. Figure 11 shows the opposite, where the output is close to the values of the accelerometer. Having a high confidence in the measurements could cause higher noise in the output of the KF.

Q_k indicates the confidence in the model derived. The elements of Q_k are assumed to be uncorrelated, and therefore it is a diagonal matrix, composed of Q_{angle} (covariance of the angle) and Q_{bias} (covariance of the bias).

$$Q_k = \begin{bmatrix} Q_{angle} & 0 \\ 0 & Q_{bias} \end{bmatrix} \quad (31)$$

The effects of changing Q_{angle} are opposite to R_k and Q_{bias} is adjusted until no drift is observed in the Kalman Filter output. Figures 12 and 13 show the influence of Q_{angle} in the output of the KF. Figure 14 shows the KF running for about 40 seconds, and it can be seen that the output has not deviated from the average of the accelerometer reading, suggesting that the Q_{bias} is appropriate.

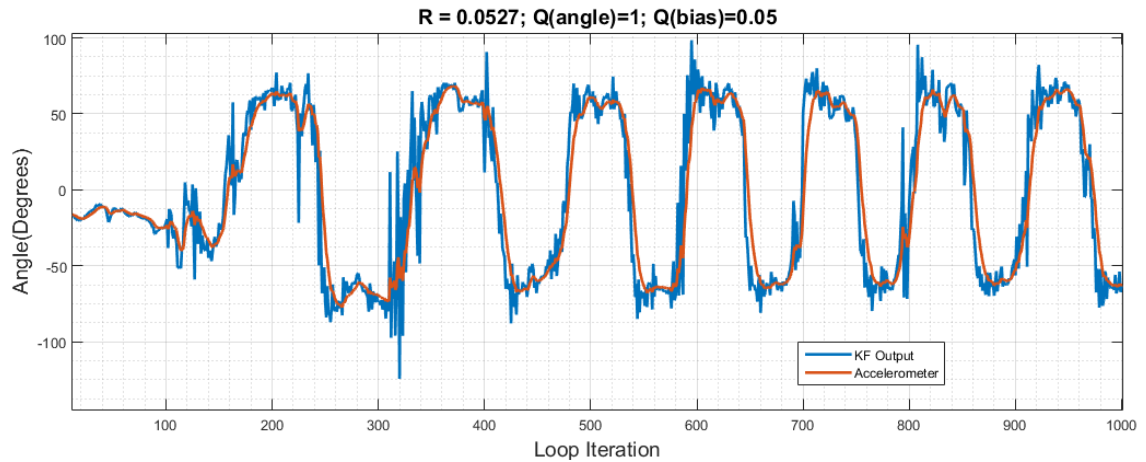


Figure 12: Graph Showing the effect of high Q_{angle}

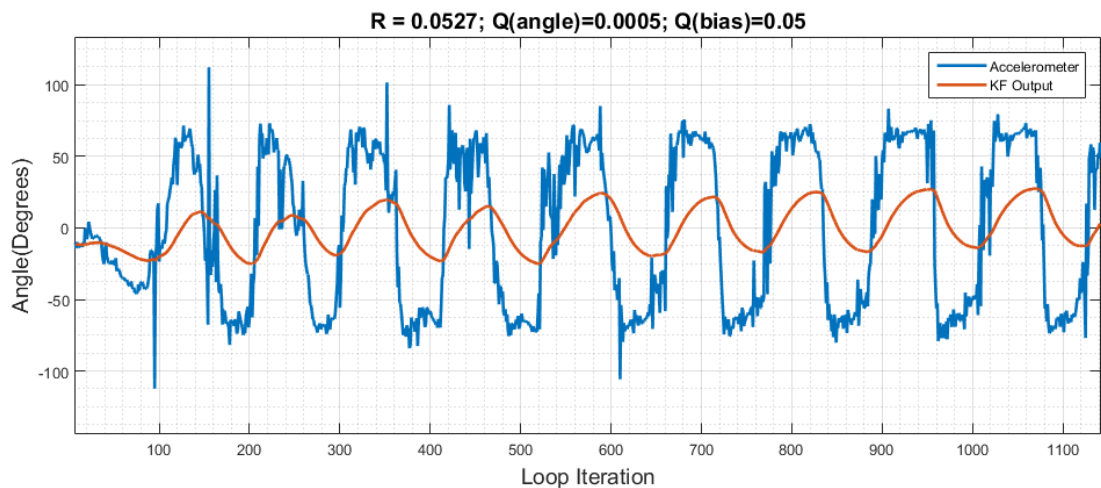


Figure 12: : Graph Showing the effect of low Q_{angle}

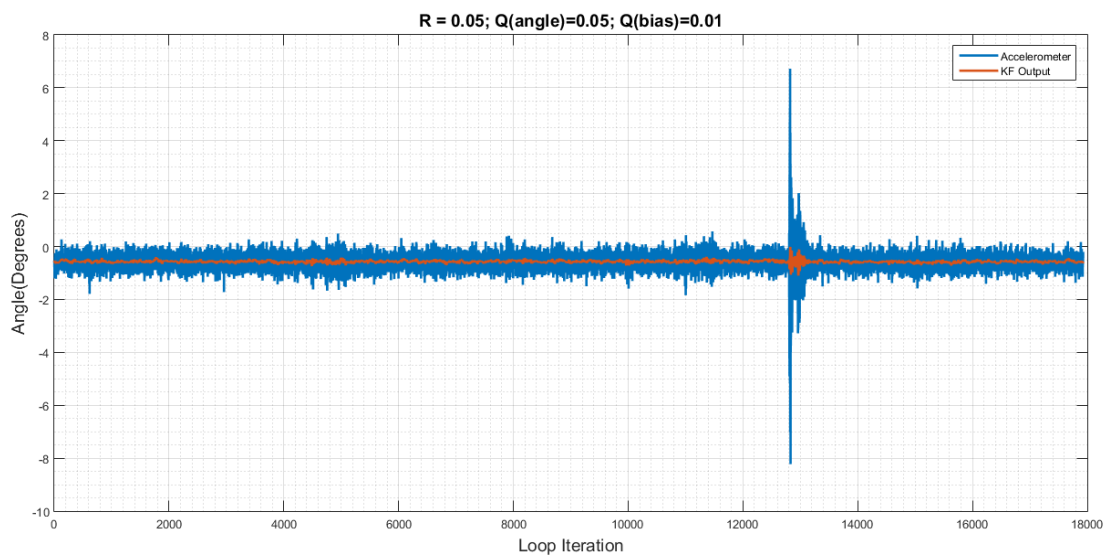


Figure 14: Graph showing that the angle is the appropriate value

3.5. Complementary Filter

This section is based on the work by Esfandyari et al [22] . The diagram below outlines the complementary filter (CF) for fusing data from a gyroscope and an accelerometer:

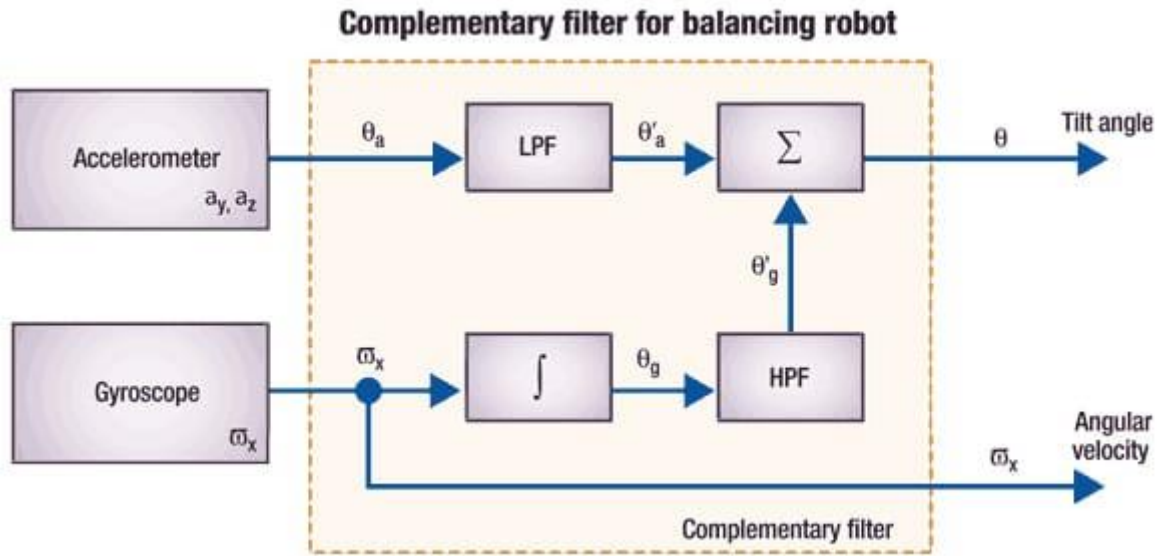


Figure 13: Diagram of the Complementary Filter [22]

The diagram can be translated into the equation below:

$$\theta = \beta \cdot \theta_{gyro} + (1 - \beta) \cdot \theta_{accelerometer} \quad (32)$$

Where β is a constant between zero and one. θ_{gyro} is calculated using equation (7) and the accelerometer using equation (23). The bias is estimated in the start up sequence, the same way it is estimated for the KF.

In essence, the first part of the equation behaves like a high pass filter and the second like a low pass filter. β is generally set to a relatively high value (close to 1), it is set depending on the sampling rate(ΔT) and time constant(τ) desired. τ is given by

$$\tau = \frac{\beta \cdot \Delta T}{1 - \beta} \quad (33)$$

τ affects the weighting in equation 31. If the motion of is faster than τ , higher weighting is placed upon the gyroscope angle, behaving as the low pass filter and reducing the noise from the accelerometer. The opposite happens when the motion is slower than τ , the accelerometer angle measurement has a higher weighting decreasing the effect of the bias in the gyroscope.

3.6. Analysis of Performance

To test the performance of the filtering techniques and to tune the Kalman Filter the test rig shown in Figure 14 was designed and made. The length of the arm, which the IMU is secured to, was to mimic the position of the IMU in the robot. Initially, the servo motors where programmed to move the arm, however, the servo motors are not sufficiently fast, to notice any delay in the output of the sensor fusion techniques.

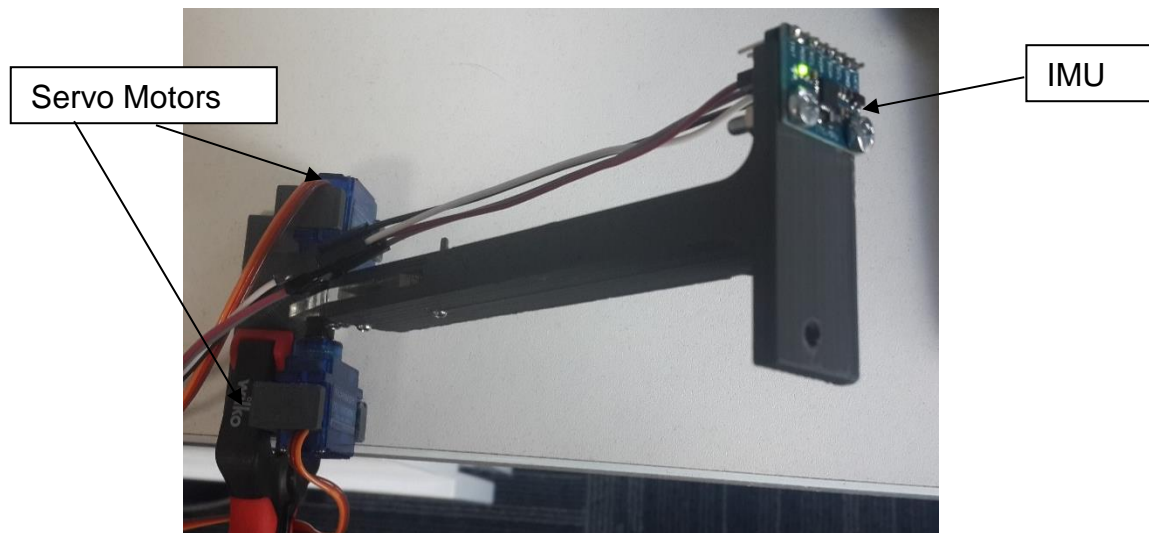


Figure 14: Picture of the Test Rig

The rig was still useful as it holds the position after no external force is applied. By manually moving the arm, the waveforms in Figures 10-14 were generated. The final tuning parameters for the KF are shown in Figure 15 and it can be observed that with these values the output converges sufficiently fast whilst the accelerometer noise is filtered out.

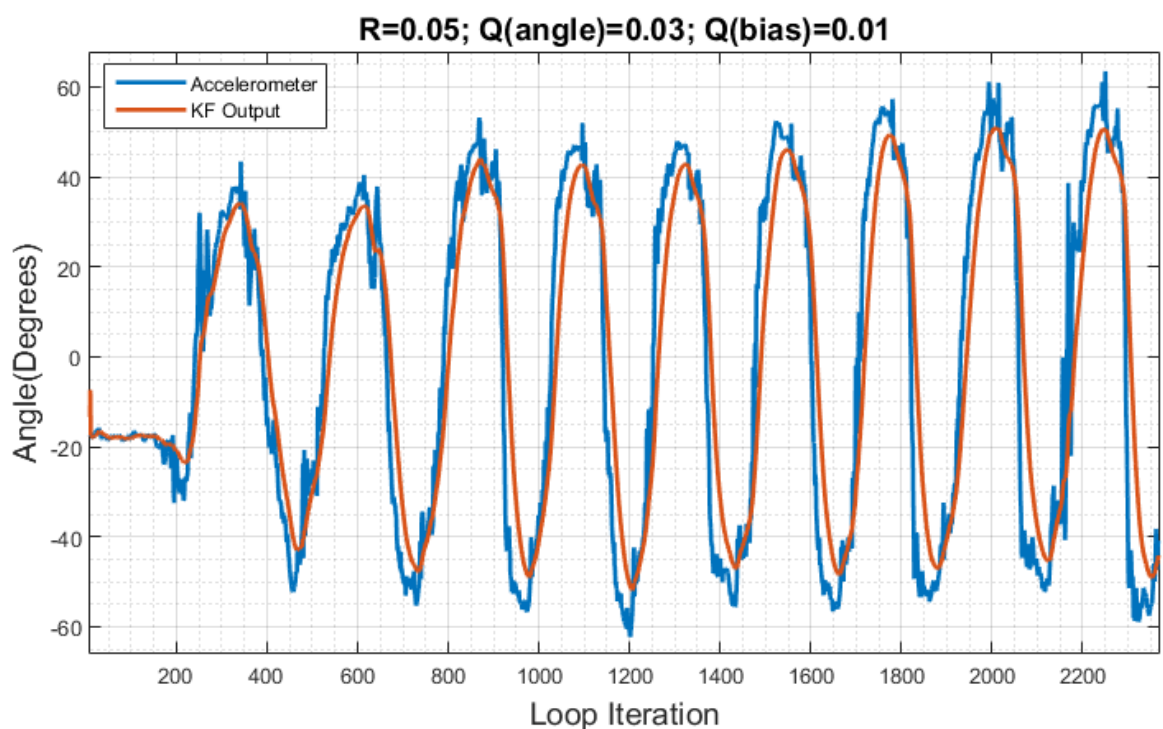


Figure 15: Graph showing the performance of the KF after being tuned

As a final comparison, the output of the Kalman Filter, the Complementary Filter and the built in algorithm was plotted in Figure 16.

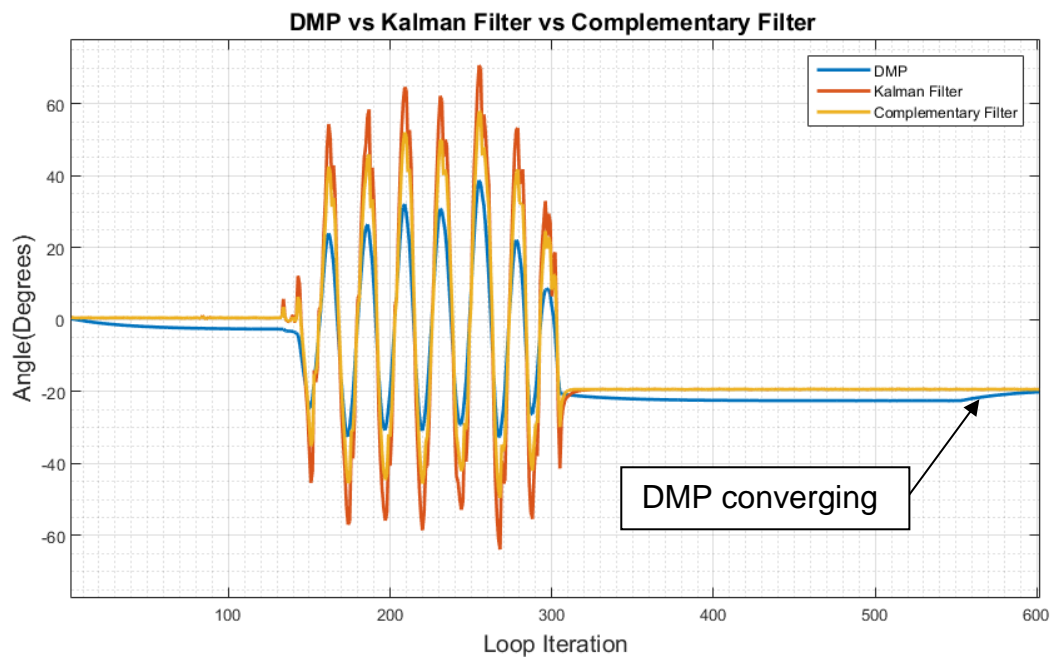


Figure 16: Graph showing the output angles of the DMP, KF and CF.

As the manufacturers of the chip developed the built in Digital-Motion Processor (DMP), it was expected to have the best performance and was going to be used as the benchmark for comparison of the different filtering techniques. In practice, Figure 16 shows that the DMP takes longer to settle than the Kalman Filter and the Complementary Filter. It may be because the frequency had to be lowered to 20Hz otherwise the buffer in the IMU would overflow. The Kalman Filter and the Complementary Filter appear to have very similar performance, the only further comparison required would be to observe whether the complementary filter would drift over time due to the increasing bias.

Lastly, the table below shows loop execution time of each of the angle estimation methods and it can be seen that the KF only requires an additional 270 microseconds to complete the calculations in comparison to the Complementary Filter, making no significant impact on the overall loop execution time.

Angle Estimation Method	Average Loop Iteration time (μ S)
Kalman Filter	1740
Complementary Filter	1470
DMP	1462

Table 2: Average Loop Time of Each of the Fusion Techniques

4. Hardware

4.1. Introduction

The section aims to provide an overview of the design considerations and implementation of the robot physically. To begin with, the torque required by the motor is estimated and the compromise between torque and RPM is discussed. Followed by a description of the main components such as the microcontroller, the IMU, the motor driver board and the power source. To finalise, the overall design of the robot is shown.

4.2. Motors and Wheels

To determine the appropriate motors for the robot, the first consideration was the minimum torque required. To estimate the torque, the model to the right was considered. The relationship between torque, τ , and force, F , is given by

$$\tau = \|\mathbf{r}\| \|\mathbf{F}\| \sin \theta \quad (34)$$

Where \mathbf{r} is the position vector and θ is the angle between force and position vectors.

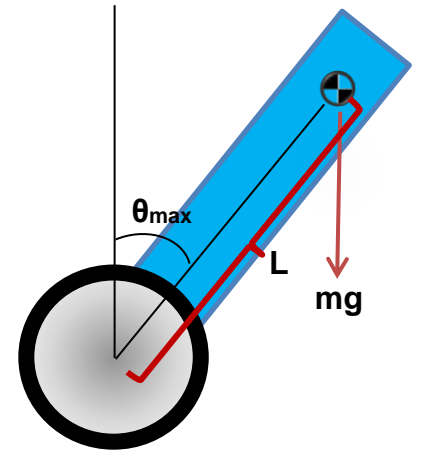


Figure 17: Robot Model

Assuming the distance between the pivot point and the centre of mass (L) is 12cm, the maximum tilt angle (θ_{\max}) is 40° and the mass of the robot (m) is 0.7kg, the minimum torque required is

$$\tau = L * mg * \sin \theta = 0.12 * 0.7 * 9.81 * \sin(40) = 0.530 \text{ Nm} \quad (35)$$

ven that there are two motors, the torque required per motor is 0.265Nm. In this scenario, the inertia is not taken into account and the robot is assume to start moving at θ_{\max} . Due to the crude assumptions, the robot by Gornicki was looked at [16]. In which the stall torque of the motors were 0.224Nm and a gear with a 3:1 ratio was used. With the assumption of 15% inefficiency, that corresponds to a maximum torque on the output shaft of 0.5712Nm.

Looking solely at the parameters mentioned, the chosen motor was the Pololu medium power 47:1 Metal Gearmotor with 48 CPR Encoder. The motor has a stall torque of 0.611Nm and the encoder outputs 2248.86 counts per revolution [34]. However, this was not the appropriate manner to choose the motors. The wheels and

the RPM also have to take into consideration. The torque required to move the wheel is also given by Equation X, where r would be the radius of the wheel and θ would be 90 degrees, thus for a given force the torque is proportional to the wheel radius. The trade-off of choosing smaller wheels such that lower torque is required is lower circumference of the wheel, requiring the motors to spin faster to get under the centre of mass of the robot.

The dispute between torque and RPM is also present with the distance between the centre of mass and the pivot. As shown by Equation X, as L increases, the torque required also increases, on the other hand the natural frequency, ω_p of the pendulum [35], is given by

$$\omega_p = \sqrt{\frac{g}{L}} \quad (36)$$

Increasing L has the effect of reducing the natural frequency of the system, decreasing how fast the actuators have to perform.

As mentioned, unfortunately the natural frequency and the wheel size were only considered after motor was purchased. To attempt to improve the system's performance the battery was moved to higher position to increase the centre of mass. In addition, the wheels were changed from 60mm to 80mm in diameter.

4.3. Microcontroller

The microcontroller used in the robot is the Arduino Uno. It is a board based on the ATmega328P from Atmel's AVR family. It is an 8-bit microcontroller running with a 16 MHz clock speed. The board has a built-in voltage regulator allowing it to be powered by any input voltage in the range of 6-20V. On-board is also an FTDI chip, making only a USB cable required to program it [36]. The board has a relatively small size, maintaining the robot as small as possible. The main advantage of the Arduino is the IDE and the large community. Its IDE enables fast software development due to the extensive collection of libraries and sample code. The large community is helpful in the case where a problem is encountered, there is a higher chance that someone else has found a solution and it is visible in one of the many forums available.

4.4. Inertial Measurement Unit (IMU)

The inertial measurement unit (IMU) is very important component in the robot as knowing the tilt angle is critical. IMUs are composed of electromechanical systems (MEMS). MEMS accelerometers and gyroscopes have the advantage of being compact, inexpensive and having low power consumption. They are however less accurate in comparison to optical devices [25].

The IMU used is InvenSense's MPU6050. The specifications mentioned of the IMU in this section, can be found in the data sheet [37]. It is a 6 degrees of freedom IMU, consisting of a 3-axis gyroscope and a 3-axis accelerometer. Initially the IMU used was the MPU9250, however due to malfunction, the sensor was replaced. These sensors are very similar, with the exception of the magnetometer found on the MPU9250. A great advantage of the sensor is that it has been used with an Arduino and libraries are available for it.

The gyroscope full-scale range that can be adjusted to ± 250 , ± 500 , ± 1000 or ± 2000 degrees per second. The gyroscope was set up to have the range between ± 1000 degrees per second, with these settings the gyroscope will have the lowest resolution but looking at equation 36 and assuming $L = 0.12m$, the natural frequency of the system is 9.04 radians/second (corresponding to 517 degrees per second), any lower range would cause the motion not to be detected. The gyroscope also has an adjustable sampling frequency between 1 to 8 kHz. Upon brief experimentation, it was concluded that the set sampling frequency did not have a significant impact on the results; it may be because the microcontroller only read data from the sensor at 400 Hz.

The accelerometer can also be programmed to have different full-scale ranges, these include to ± 2 , ± 4 and $\pm 6g$. The range chosen is $\pm 2g$ as the assumption while calculating the angle from the accelerometer is that gravity is the only force acting upon the robot. The MPU6050's accelerometer has a sampling rate of 1 kHz.

The communication between the microcontroller and the IMU is through the Inter-Integrated Circuit (I2C) protocol. The protocol is used for set-up of the IMU and reading data from it. Further details on the communication and set-up of the IMU are provided in the Section 5.

4.5. Motor Driver Board

The motors chosen are designed to operate at 12V and have a stall current of 2.1 Amps. The microcontroller cannot supply that much power, thus a Full bridge driver is required to allow the motor to be controller in both directions. The motor driver board used is the Aptinex Dual L6203 Board. The board itself has diodes to protect the microcontroller and battery from back EMF. The L6203 chips can operate with a supply voltage up to 42V, and can supply an RMS current of 4A (peak 5A). They are also compatible with TTL logic, meaning that the microcontroller can be connected directly to the control pins without the need of transistors in between. Lastly, the enable pins can be controller with a Pulse Width Modulation (PWM) frequency of up to 100 KHz [38], at higher frequencies the motor was found to have higher torque it may be due to higher current supplied.

4.6. Power Source

To provide power while maintaining the robot mobile, a Lithium Polymer (Li-Po) battery was chosen as the power source. The specific battery used is Hobbyking's Turnigy 3 cell 2200mAh 20C. The battery can supply a current of up to 44A. Given that the stall current of each of the motors is 2.1A and the remaining components (Arduino, IMU and encoders) have an estimated current draw of 500 mA, the battery can effortlessly manage [34]. The large current can be dangerous for the components, thus a fuse had to be put in place.

4.7. Overall Design

The final design of the robot is shown in the following page. The modular design allowed the layers to be adjusted as needed, as mentioned, the positioning of the battery was moved higher up to reduce the natural frequency of the system. The heights were also adjusted, being composed of M3 spacers, it simply involved adding a few more in between the layers. The layered format also kept the components protected, this was particularly important with the battery as a puncture or dent on a Li-Po battery may cause an explosion.

Initially, the layers were planned to be made out of MDF, however it was not available, they were instead made out of Acetal. Acetal can be laser cut and is easy to work with. Furthermore, the material has a high tensile strength without being brittle or dense, this means that it will not be excessively heavy and can withstand all the drops while the robot is being tuned.

Additionally, potentiometers and a switch were added. The potentiometers are to help tune PID and the switch is to allow the power to be cut off without having to unplug the battery cable every time. To hold them in place the following pieces were designed and 3D Printed:

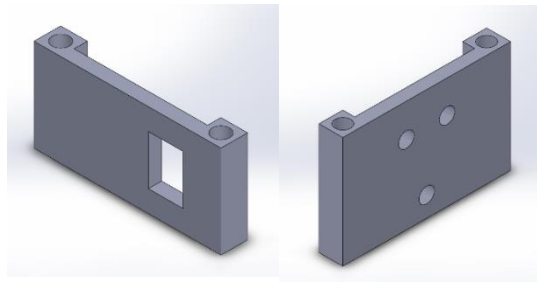


Figure 18: 3D Printed Parts

4.8. Final Assembly

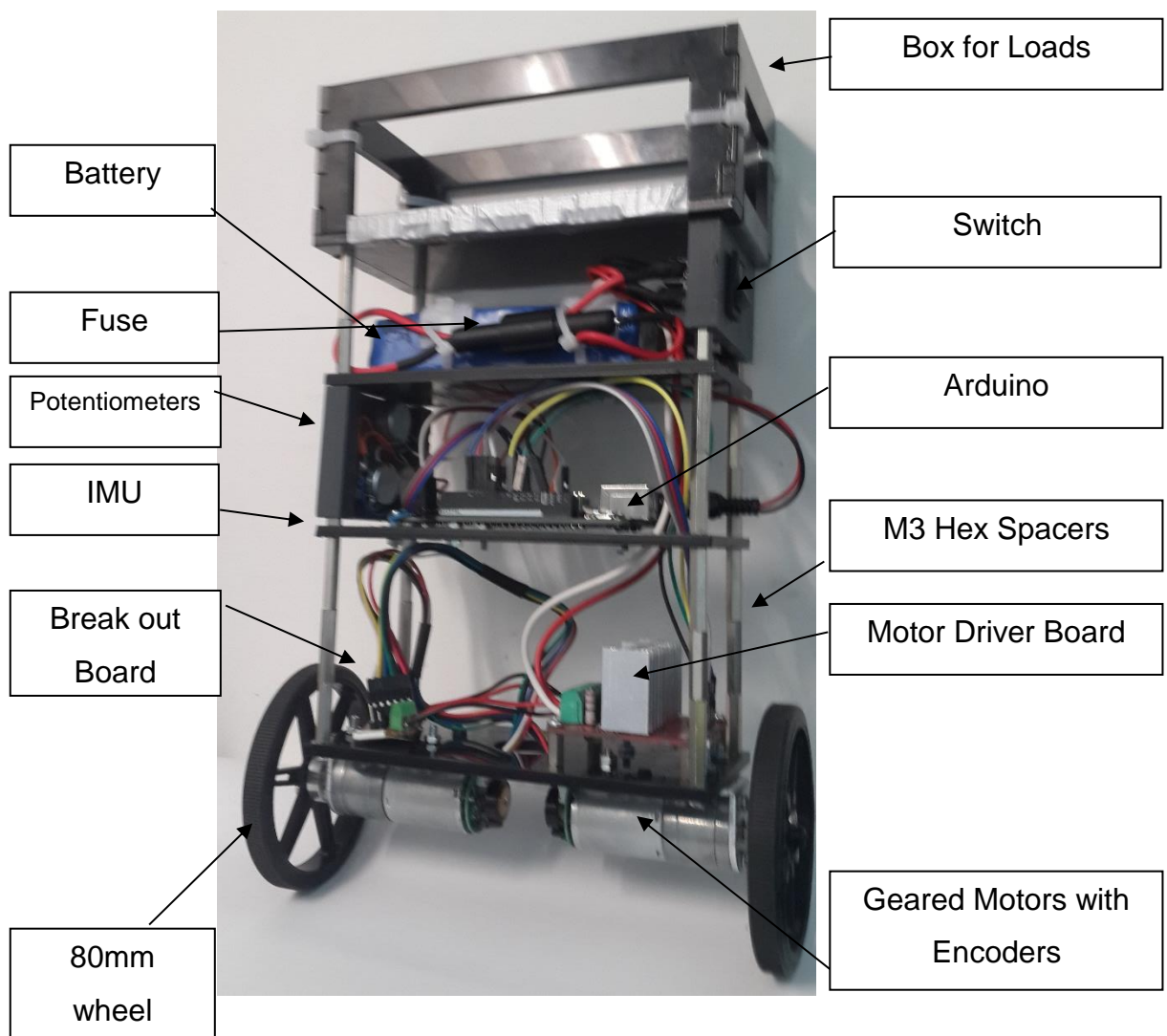


Figure 19: Final Robot Assembly

5. Software Implementation

5.1. Introduction

This section provides in detail the key aspects of programming the robot. The code is composed of various sections found online, this maybe in the form of forums, blogs or tutorials. Throughout the project various versions of code developed, each for a specific purpose, for example just printing the output of the Kalman Filter and the accelerometer angle to view the performance of the Kalman Filter or code to characterize the motors. Part of the code is the same as it would be in an equation from, thus only the non-trivial aspects are considered here and the reasons behind are mentioned. To begin with, the important sections of the code are mentioned. Later, the block diagram of the code is given to provide a higher level view of the code.

5.2. Constant Loop Time

A large number of the equations mentioned in the previous sections have dt incorporated in them. Either dt can be measured or the microcontroller could be programmed to have a constant loop time. At first interrupts were proposed, however interrupts are required for encoders and prioritisation of interrupts would have to be considered. To solve this, the loop was kept constant using the *Micros()* function which when called returns the number or microseconds the Arduino has been running since being turned on. The *Micros()* is evaluated in a while loop condition, to do nothing, until the specified time is met. To test if the loop time was constant, the loop time was set to 2.5ms and a pin was programmed to remain HIGH for 1ms and then go LOW. The pin was then connected to an oscilloscope and the output is shown in the figure below. As can be seen this worked as expected:

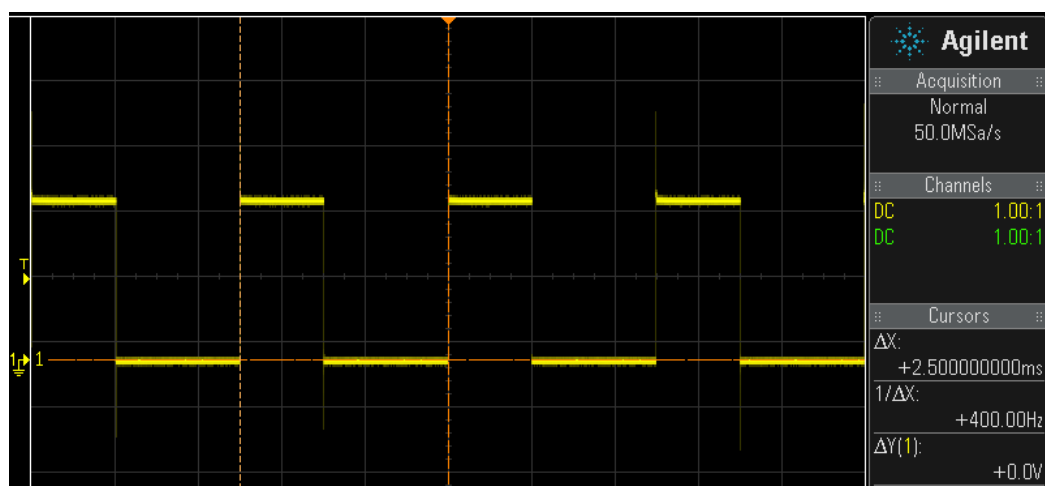


Figure 20: Oscilloscope Output

5.3. Reading from IMU

As mentioned previously, the communication between the IMU and the microcontroller is done through the I2C protocol. The protocol involves a series of rules, such as the master (Arduino) driving the clock line or the master initiating transfer of data. These are not of concern, as the *Wire.h* library accommodates for the rules. However, the format in which reading and writing of data is done, still needs to be followed [26]. This has been simplified and summarized in the table below:

<u>Read</u>	<u>Write</u>
1. Send the start sequence, specifying the address of the slave device and the write bit HIGH	1. Send the start sequence, specifying the address of the slave device and the write bit HIGH
2. Send the internal register address that writing is done to	2. Send the internal register address that is going to be read from
3. Send data byte	3. Send the start sequence again, specifying the address of the slave device and the write bit LOW
4. Send stop sequence	4. Read data from the register
	5. Send stop sequence

Table 3: I2C Read/Write Protocol

The device address and the register addresses can be found in the register map [39]. The register map also the values to write to configure the device as desire for it to operate, for example writing 0x00 to register 0x1C to set the accelerometer full scale to $\pm 2\text{gs}$.

Once the IMU has been set up and the reading is done from the appropriate registers, the values have to be scaled to a unit that is meaningful. The data read is the value from the Analogue to Digital Converters (ADCs) in the IMU, to convert the following formula can be used:

$$Scaled\ Value = \frac{ADC_{value}}{ADC_{resolution}} \times ScaleSize \quad (37)$$

The ADCs in the IMU have a 16-bit resolution and the scale size is as configured [26].

5.4. KF Implementation

From the Kalman Filtering Section, it can be seen that the algorithm involves matrix multiplication, addition and inversion, to do so the *MatrixMath.h* library could be used, however this would pose a significant delay in the loop iteration time. To solve the issue, the matrices have been expanded and simplified before writing the code. For example, equation X can be written as follows:

$$K_1 = \frac{P_{11}}{P_{11} + R} \quad (38)$$

$$K_2 = \frac{P_{21}}{P_{11} + R} \quad (39)$$

Separating the matrix into two sets of equations removes the need to use the library. Furthermore, changing the inverse to division will decrease the computation required by the microcontroller as the library would have used the Gauss-Jordan elimination with partial pivoting. Changing inversion to multiplication is only allowed in this case, as the values inside the parentheses simplify into a scalar. Making such changes is important with the Arduino Uno as it is an 8-bit microcontroller and has no floating-point unit.

5.5. Displacement and Angular Velocity from Encoders

The encoders in the motors are quadrature encoders. There are two signals and the rising edges are counted to establish the displacement and angular velocity. The direction in which the shaft is rotating is determined by the phase difference between the two signals. This is shown in the diagram below:

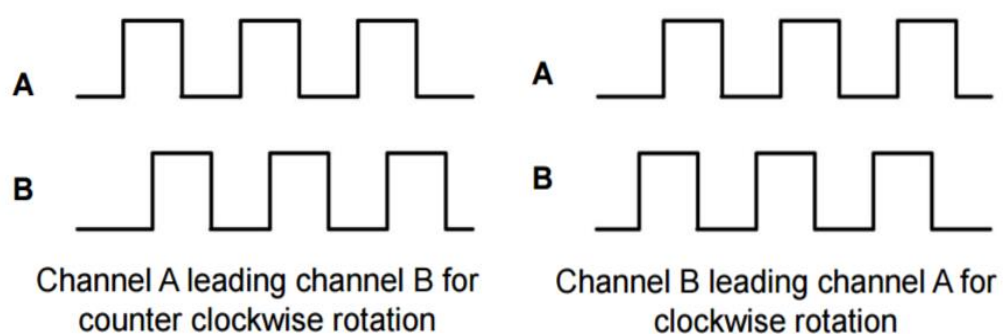


Figure 21: Shows the direction is determined with quadrature encoders [41]

To count the edges and determine the phase different, rising edge interrupts were setup on channel A. In which if, channel B is HIGH then increment the number of pulses or if B is LOW decrement the number of pulses. This method simplifies the

code and reduces the number interrupt pins needed, however, it increases the quantisation error as the number of ticks per revolution falls from 2248 to 562.

To convert the number of pulses, N , to rotor displacement and angular velocity the following formulae are used:

$$\text{Rotor Displacement [degrees]} = \text{edge count} \cdot \frac{360}{N_{PPR}} + \alpha_0 \quad (40)$$

$$\omega_{shaft} \left[\frac{\text{rad}}{\text{s}} \right] = \frac{2\pi \cdot N}{N_{PPR} \cdot T} \quad (41)$$

Where N_{PPR} is the number of pulses per revolution of the encoder, α_0 is the initial position (assumed to be zero), and T is the time window between readings. The quantisation error is also inversely proportional to T . To reduce the quantisation error, the time window was set to 10 loop iterations.

5.6. Overall Diagram

The complete software loop is given in the following page. Most of the code is sequential with the only exception being the encoder interrupts, as these are asynchronous. In the diagram, the interrupt is shown as a grey box, with grey arrows to and from it. The dashed arrows show that it can happen anywhere in the code, at any given moment.

The initialisation involves the following: setting up the serial communication with the computer, setting up the IMU, declaring the input and output pins, setting the PWM frequency, and estimating the bias and initial position for the Kalman Filter.

The output to the motor driver board simply involves setting which part of the H-bridge is going to be on, to have the correct direction of motion of the wheels. In addition, setting the PWM duty cycle for the appropriate angular velocity of the wheels.

Code Block Diagram

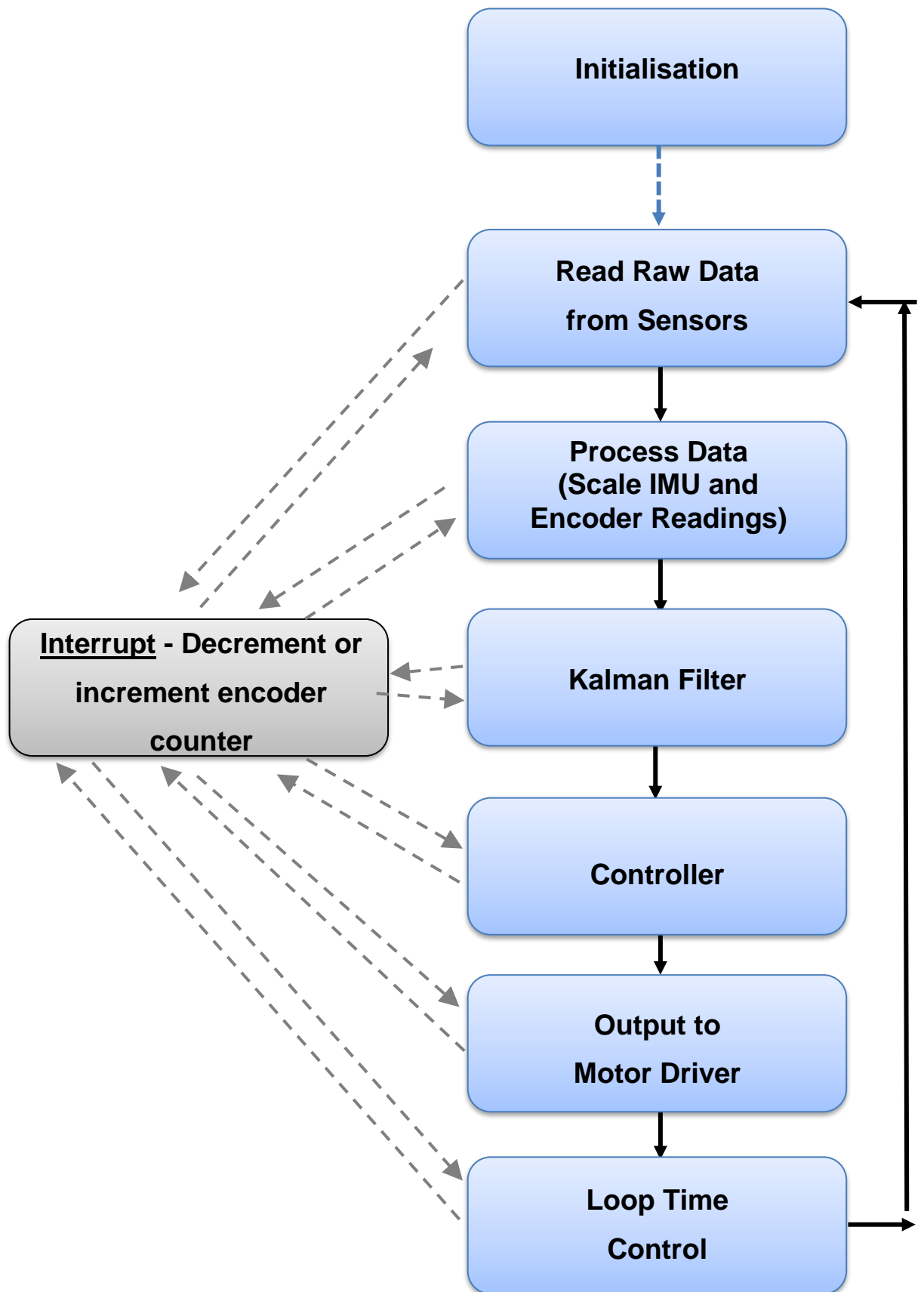


Figure 22: Code Block Diagram

6. Control

6.1. Introduction

This chapter focusses on the control aspect of the self-balancing robot designed. Firstly the state space model of the robot is shown and the how the parameters were established. Then a discussion on stability of the plant and control of the plant in simulation. Finally, the robot's physical performance is shown and analysed.

6.2. State-Space Model

Developing the model is a tedious task, thus the model developed by Bonafilia et al was used. The model has been developed from the three models shown below:

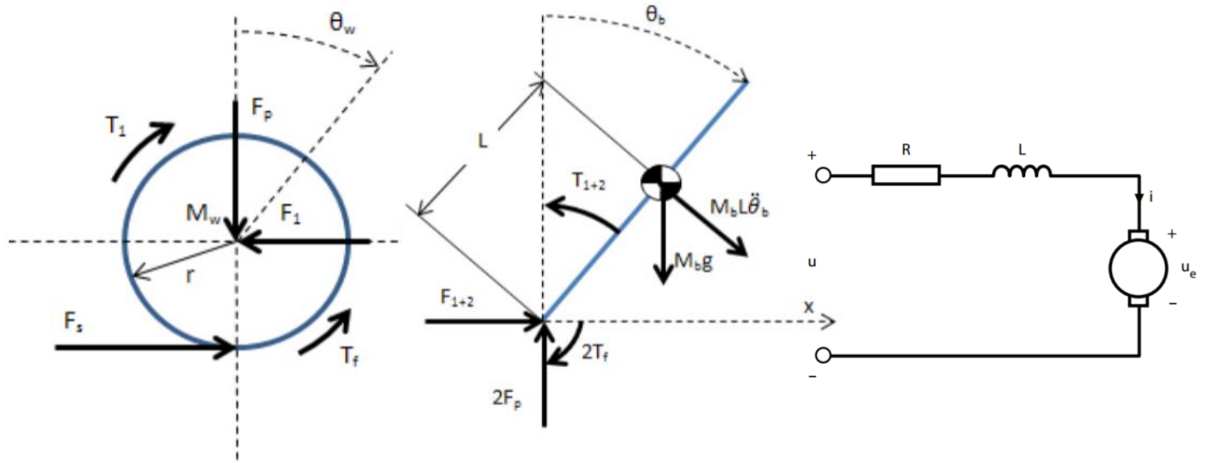


Figure 23: [a] diagram of force and torque on the wheels, [b] diagram of upper body of the robot, [c] diagram of electrical sub-system [14]

The model is based upon the following assumptions: No yaw is considered, the body and wheel are represented by point masses, no slipping between the ground and wheels, and the mechanical system is slower than the electrical system [17].

The linearized model given in the following state-space format

$$\dot{\xi} = A\xi + Bu \quad (42)$$

$$y = C\xi + Du \quad (43)$$

With the following states:

$$\xi = [x \quad \dot{x} \quad \theta \quad \dot{\theta}] \quad (44)$$

Where x is the displacement, \dot{x} is the velocity, θ is the tilt angle and $\dot{\theta}$ is the angular velocity. The Matrices A , B , C and D are shown below and the parameters are given in table 4.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \alpha & \beta & -r\alpha \\ 0 & 0 & 0 & 1 \\ 0 & \gamma & \delta & -r\gamma \end{bmatrix} \quad (45)$$

$$B = [0 \quad \alpha\varepsilon \quad 0 \quad \gamma\varepsilon]^T \quad (46)$$

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (47)$$

$$D = [0 \quad 0]^T \quad (48)$$

Where

$$\alpha = \frac{2(Rb - K_e K_m)(M_b L^2 + M_b r L + J_b)}{R(2(J_b J_w + J_w L^2 M_b + J_b M_w r^2 + L^2 M_b M_w r^2) + J_b M_b r^2)} \quad (49)$$

$$\beta = \frac{-L^2 M_b^2 g r^2}{J_b(2J_w + M_b r^2 + 2M_w r^2) + 2J_w L^2 M_b + 2L^2 M_b M_w r^2} \quad (50)$$

$$\gamma = \frac{2(Rb - K_e K_m)(2J_w + M_b r^2 + 2M_w r^2 + L M_b r)}{Rr(2(J_b J_w + J_w L^2 M_b + J_b M_w r^2 + L^2 M_b M_w r^2) + J_b M_b r^2)} \quad (51)$$

$$\delta = \frac{L M_b g(2J_w + M_b r^2 + 2M_w r^2)}{2J_b J_w + 2J_w L^2 M_b + J_b M_b r^2 + 2J_b M_w r^2 + 2J_b M_w r^2 + 2L^2 M_b M_w r^2} \quad (52)$$

$$\varepsilon = \frac{K_m r}{Rb - K_e K_m} \quad (53)$$

Parameter	Value	Description
M_b	0.987	Mass of the robot [kg]
M_w	0.025	Mass of the wheels [kg]
J_b	3.83e-3	Moment of inertia about the centre of mass [kgm ²]
r	0.04	Radius of the wheels [m]
J_w	4e-05	Moment of inertia for the wheels [kgm ²]
L	0.102	Distance of wheel to centre of Mass
K_e	0.855	EMF constant [Vs/rad]
K_m	0.316	Torque constant [Nm/A]
R	7.2	Motor resistance [Ω]
b	0.002	Viscous friction constant [Nm s / rad]
g	9.81	Gravitational constant [m / s ²]

Table 4: Parameters of the robot

6.3. Estimation of the Robot's Parameters

Estimation of some of the parameters is simple, such as the mass of the robot using a scale or the resistance of the motors using a multimeter. The viscous friction constant was assumed to be the same as the one found by Bonafilia et al. The inertia value for the wheel was calculated by assuming the wheel is solid cylinder rotating around its centre and using the formula below [40]:

$$J_w = \frac{1}{2} M_w r^2 = \frac{1}{2} (0.025 \times 2) (0.04)^2 = 4 \times 10^{-5} \text{ kgm}^2 \quad (54)$$

The robot's body was assumed to be a homogenous parallelepiped and was calculated using the formula below, where h is the height and w is the width:

$$J_b = \frac{M_b}{12} (h^2 + w^2) = \frac{0.987}{12} (0.204^2 + 0.07^2) = 3.83 \times 10^{-3} \text{ kgm}^2 \quad (55)$$

The torque constant, K_m , is 1/gradient of the current/torque graph shown to the right. The values provided in the data sheet are the stall current, stall torque and the free running current. From these value K_m was found to be 0.316 Nm/A.

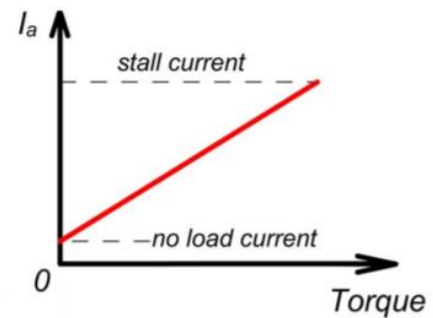


Figure 24: Current vs Torque Graph

To find K_e different voltages were applied to the motors and the values of angular velocity were plotted (Figure 26). Using the method of Least squares estimation the equation of the line of best fit was found and is given on the graph legend in the format of $y = A + Bx$.

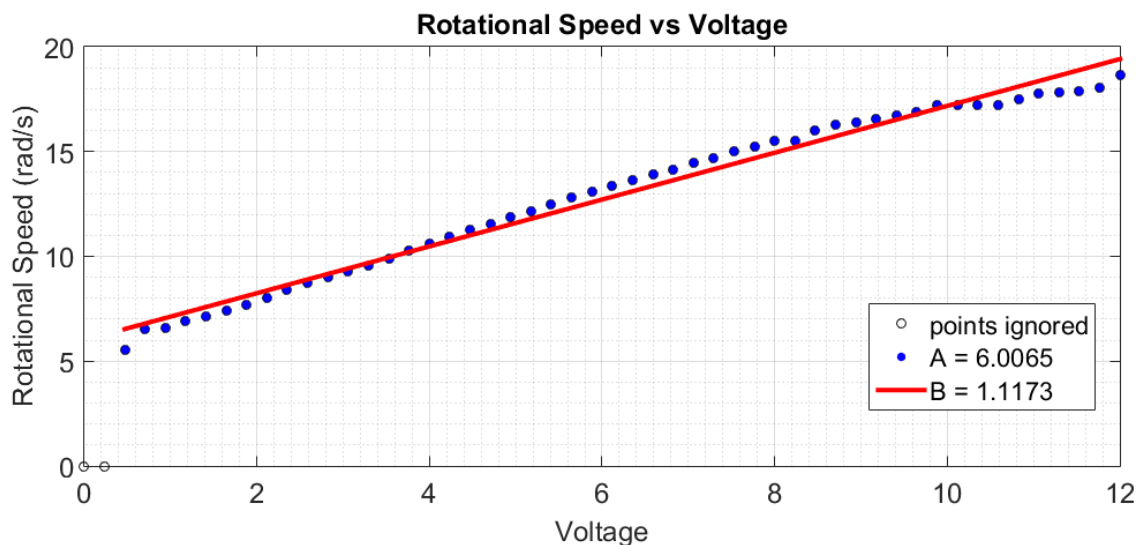


Figure 25: Graph of Speed vs Voltage

K_e is the inverse of the gradient, hence given by

$$K_e = \frac{1}{B} = 0.855 \frac{Vs}{rad} \quad (56)$$

6.4. Analysis of the System

Using the MATLAB toolbox to convert the state space model into a transfer function and pole-zero plotting function, the figure below was generated:

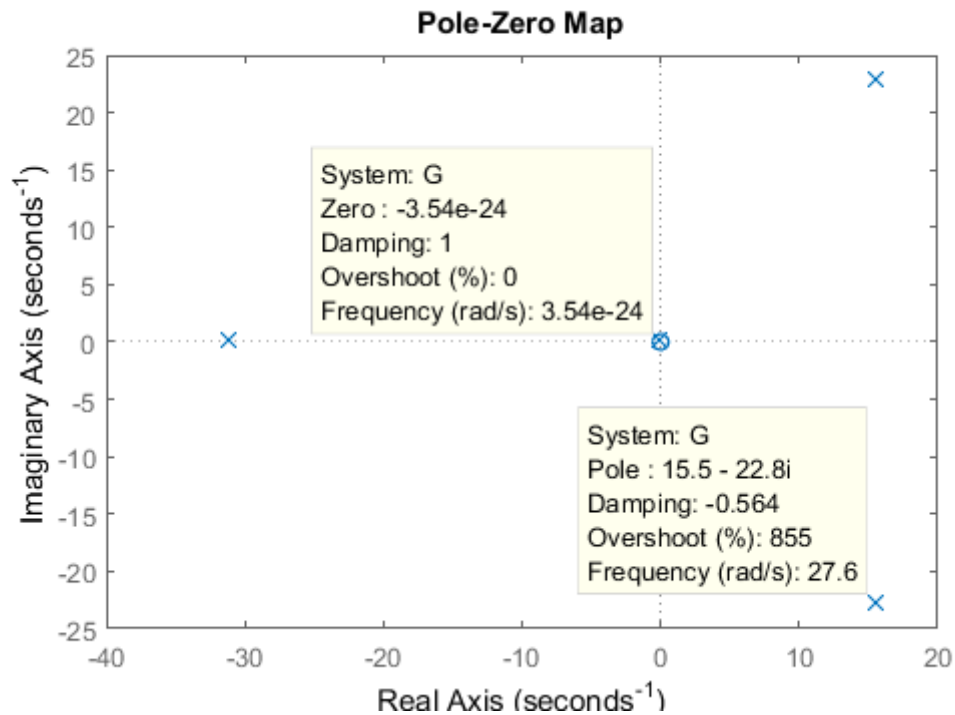


Figure 26: Pole-Zero Plot

As it can be seen, there are positive poles in the plant, once again showing that the plant is unstable. Unlike as expected, non-minimum phase zeros are not present in the system, it may be due to how the system model was derived. Furthermore, dissimilar to the inverted pendulum on a cart there are no positive real poles to the right of the non-minimum phase zero, hence a stable controller can be used to control the plant.

6.5. LQR in MATLAB

In the robot designed, all of the states can be measured thus

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (57)$$

Standard selection of Q is given by

$$Q = C^T C \quad (58)$$

Thus Q will be the same as (55). The value of R was selected arbitrarily to one.

Using the LQR function in MATLAB, the following K matrix was obtained:

$$K = [1.0000 \quad 0.1129 \quad -37.7177 \quad -1.4959] \quad (59)$$

To view the system's performance and the input signal the Simulink diagram in Figure 27 was employed. The initial conditions were set such that all the states were zero, except the tilt angle was set to 0.15 radians. The system states and the system input are shown in Figures 28 and 29 respectively.

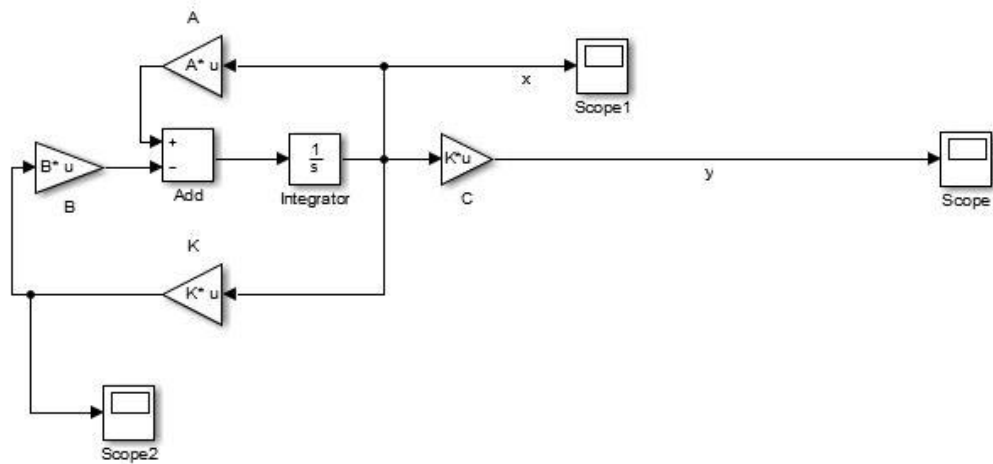


Figure 27: Simulink Diagram

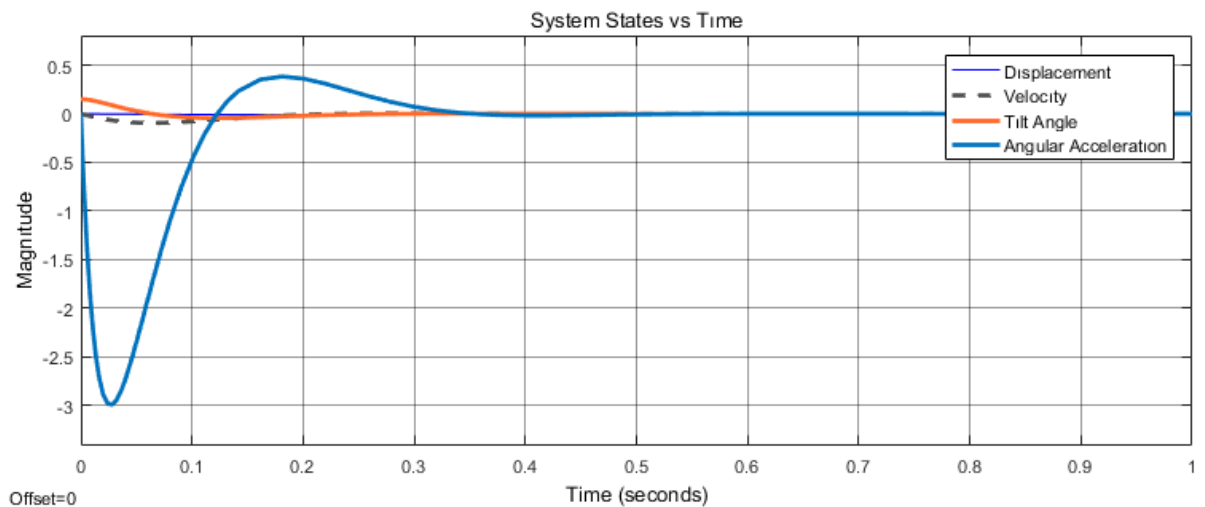


Figure 28: System states vs Time

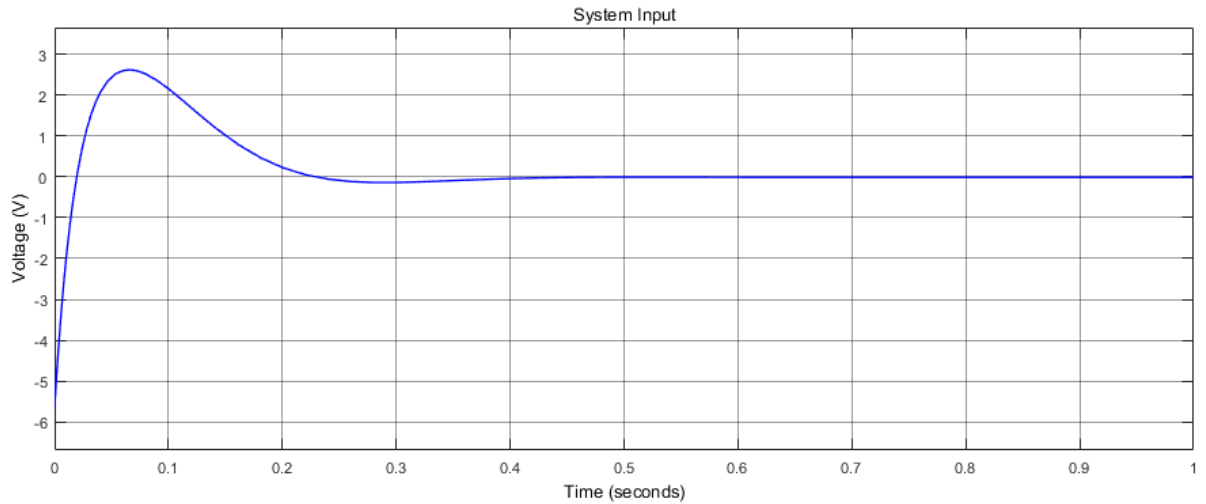


Figure 29: System Input (Voltage) vs Time

From Figure 29, it can be seen that the voltage does not exceed $\pm 12V$; hence, the calculated K values may be used in the actual robot. Figure 28 shows that the robot is expected to return to equilibrium and the initial position within a fraction of a second.

6.6. Physical Performance

Once the robot was assembled and the Kalman Filter tuned, then a PID controller was implemented. The code can be found in Appendix 1. The recorded performance is shown in the diagram below:

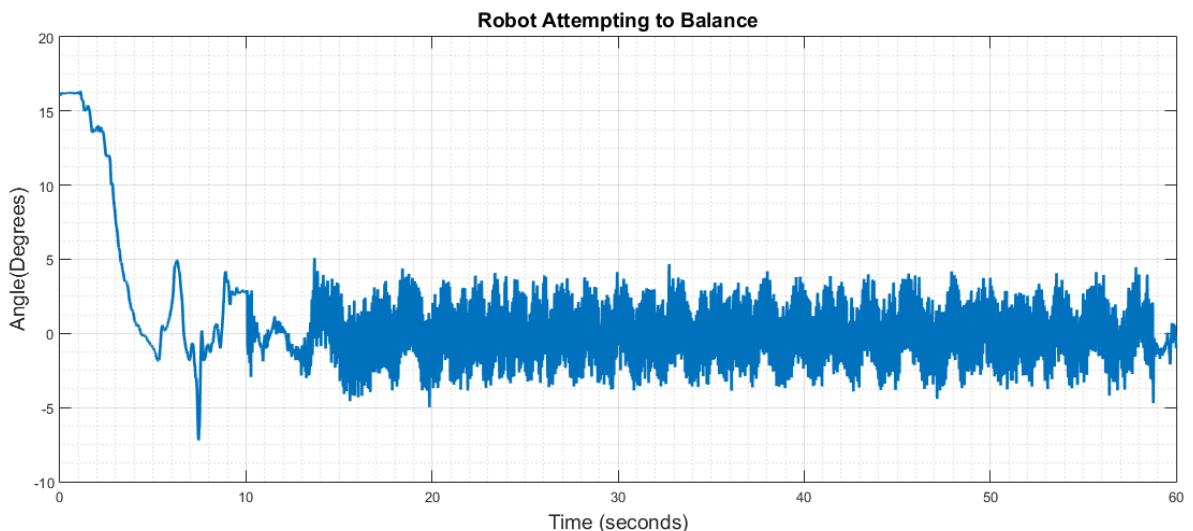


Figure 30: Robot Tilt angle

The performance is very poor; the robot was oscillating between ± 4 degrees. The values were recorded through a USB cable, which dampened the oscillations of the robot. Without the USB cable the robot will vibrate attempting to balance for a few seconds and then fall. The issue is suspected to be due to the backlash in the motors that became progressively worse as the motors were worn in, and an initial dead

zone in the Voltage against RPM graph. As an attempt to compensate, an offset was set to in the PWM duty cycle in the each given direction. According to Bonafilia et al, this problem was faced in their robot as well and an attempt to set an offset resulted in a shaky behaviour.

LQR was also implemented in the robot using the K values calculated in MATLAB, the code is given in Appendix 4. However, the robot did not maintain itself upright.

7. Conclusions

7.1. Project Achievements

The objectives set in the beginning of the project were met, with the exception of making the robot balance by itself using PID. The project as whole was a steep learning due to the wide array of disciplines involved from construction and design, to control and software implementation.

7.2. Project Limitations and Final Remarks

The robot not balancing is certainly a large limitation of the project. The focus was control and data fusion, hardware selection was rushed and an aside which caused the need to have to change battery position and the wheels. Furthermore, the motors were found to have significant backlash, which was a complaint in the manufacturer's forums. Deeper analysis could have been conducted prior to purchasing the components. Instead of using standard motors with spur gears, motors with planetary gearboxes generally tend to have less backlash. In addition, instead of the Arduino a Teensy microcontroller could have been used, the cost is similar with higher performance, and the same IDE and libraries can be used to program it. This would have resulted in less time attempting to optimise the code.

In terms of delivery, in comparison to set plan, an attempt was made to complete the targets for each given week. For example, if by week 2 Kalman tuning had to be completed and it was only finished in week 4, the targets for weeks 3 and 4 were met whilst I was trying to meet the targets for week 2. This however was also a disadvantage as it led to exploring aspects that were not included in the final robot such as LabVIEW for data visualisation or Bluetooth communication. The weeks set aside for contingency planning were very useful, major hindrance was found when the IMU stopped working.

Given more time and resources, better motors could be purchased to make the robot balance. The development of a self-balancing robot may extended in many directions for future work, the yaw can be considered, LQG controller can be implemented or remote control using Bluetooth communication.

Ultimately, it can be summarised that a learning platform was developed rather than designing and development of an optimally performing self-balancing robot.

8. References

- [1] D. Caulley, N. Nehoran and S. Zaho, "Self-balancing robot," 2015. [Online]. Available: https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/f2015/dc686_nn233_hz263/final_project_webpage_v2/dc686_nn233_hz263/. [Accessed 02 April 2017].
- [2] C. Sundin and F. Thorstensson, "Autonomous Balancing Robot," 2012.
- [3] R. C. Ooi, "Balancing a Two-Wheeled Autonomous Robot," The University of Western Australia, 2003.
- [4] O. Boubaker, "The inverted pendulum: A fundamental benchmark in control theory and robotics," IEEE, 2012.
- [5] D. Goldman, N. Gravish, S. Sharpe and H. Li, "Nonlinear Dynamics of Human Locomotion," Georgia Institute of Technology, Shanghai Jiao Tong University, 2012.
- [6] F. R, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation," IEEE SIGNAL PROCESSING MAGAZINE, 2012.
- [7] C. J, Interviewee, *5 Lines of Code to Land on the Moon*. [Interview]. 2016.
- [8] SainSmart, "SainSmart InstaBots Upright Rover Kit V2.0 Updated 2-Wheel Self-Balancing Robot Kit," [Online]. Available: <https://www.sainsmart.com/sainsmart-balancing-robot-kit-v2.html>. [Accessed 02 April 2017].
- [9] Kansas State University, "Wireless Inverted Pendulum Cart," [Online]. Available: http://www.mne.k-state.edu/static/nlc/tiki-index.php?page=S_H_WirelessInvertedPendulumCart. [Accessed 02 April 2017].
- [10] Quanser, "Linear Flexible Joint with Inverted Pendulum," [Online]. Available: http://www.quanser.com/products/linear_flexible_joint_pendulum. [Accessed 02 April 2017].
- [11] P. A. Song, "Engineering Analysis," The University of Manchester, 2016.
- [12] J. Hoagg and D. Bernstein, "Nonminimum-Phase Zeros," IEEE Control Systems Magazine, 2007.
- [13] S. A. B. Junoh, "Two-Wheeled Balancing Robot Controlled Designed Using PID," Universiti Tun Hussein Onn Malaysia, 2015.

- [14] B. Bonafilia, N. Gustafsson, P. Nyman and S. Nilson, "Self-Balancing two-wheeled robot," Chalmers University of Technology.
- [15] S. Balasubramanian and M. N. Lathiff, "Self Balancing Robot," The University of British Columbia, 2011.
- [16] K. Gornicki, "Autonomous Self Stabilizing Robot," The University of Manchester, 2015.
- [17] L. Jodensvi, V. Johansson, C. Lanfelt and S. Lofstrom, "One Robot to Roll Them All," Chalmers University of Technology, Gotenborg, 2015.
- [18] V. VanDoren, "PID:Still the One," Control Engineering, 2003.
- [19] Z. Ding, Control Systems II (EEEN30041), The University of Mancehster, 2017.
- [20] W. An and Y. Li, "Simulation and Control of a Two-wheeled Self-balancing Robot," in *IEEE Internation Conference on Rotics and Biometrics*, Shenzen, 2013.
- [21] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," 2001.
[Online]. Available:
http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf. [Accessed 01 April 2017].
- [22] STMicroelectronics, "Solutions for MEMS sensor fusion," Mouser Electronics, July 2011. [Online]. Available:
http://www.mouser.co.uk/applications/sensor_solutions_mems/. [Accessed 2017 April 02].
- [23] Bosch, "Bosch Sensortec - BNO055," [Online]. Available: https://www.bosch-sensortec.com/bst/products/all_products/bno055. [Accessed 13 April 2017].
- [24] InvenSense, "MPU-6050," [Online]. Available:
<https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/>. [Accessed 02 April 2017].
- [25] O. J. Woodman, "An introduction to inertial navigation," August 2007. [Online]. Available: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-696.pdf>. [Accessed 02 April 2017].
- [26] North Carolina State University, "Using the MPU-6050," [Online]. Available: www.cs.unca.edu/~bruce/Fall13/360/IMU_Wk8.pptx. [Accessed 02 April 2017].

- [27] Q. a. Rotations, "Jernej Barbic," [Online]. Available: <http://run.usc.edu/cs520-s12/quaternions/quaternions-cs520.pdf>. [Accessed 02 April 2017].
- [28] Freescale Semiconductor, Inc., "Tilt Sensing Using a Three-Axis," March 2013. [Online]. Available: <http://www.nxp.com/assets/documents/data/en/application-notes/AN3461.pdf>. [Accessed 02 April 2017].
- [29] Rutgers School of Arts and Sciences, "Sensing your orientation: how to use an accelerometer," [Online]. Available: <http://physics.rutgers.edu/~aatish/teach/srr/workshop3.pdf>. [Accessed 02 April 2017].
- [30] R. Faragher, "Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation," *IEEE Signal Processing Magazine*, no. September, pp. 128-132, 2012.
- [31] S. Watson and J. C. Gomez, *Mobile Robots and Autonomous Systems Part II: Estimation for Localisation and Mapping*, The University of Manchester, 2017.
- [32] J. C. Gomez, Interviewee, *Weekly Tutorials*. [Interview]. 20 February 2017.
- [33] MathWorks, "Covariance," [Online]. Available: <https://uk.mathworks.com/help/matlab/ref/cov.html#bumju45-6>. [Accessed 03 April 2017].
- [34] A. Gafar, "Self Balancing Robot: Progress Report," The University of Manchester, 2016.
- [35] D. A. Russel, "The Simple Pendulum," [Online]. Available: <http://www.acs.psu.edu/drussell/Demos/Pendulum/Pendula.html>. [Accessed 23 April 2017].
- [36] Arduino, "Arduino Uno & Genuino Uno," [Online]. Available: <https://www.arduino.cc/en/main/arduinoBoardUno>. [Accessed 22 April 2017].
- [37] InvenSense, "MPU-6000 and MPU-6050 Product Specification Revision 3.4," 08 August 2013. [Online]. Available: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf>. [Accessed 02 April 2017].
- [38] STMicroelectronics, "L6201 - L6202 - L6203 - DMOS FULL BRIDGE DRIVER," [Online]. Available: <http://www.st.com/content/ccc/resource/technical/document/datasheet/03/af/9d/d5/a2/56/46/f6/CD00000089.pdf/files/CD00000089.pdf/jcr:content/translations/en.CD00000089.pdf>. [Accessed 17 April 2017].

- [39] InvenSense, "MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2," 19 08 2013. [Online]. Available: <https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf>. [Accessed 07 April 2017].
- [40] Hyper Physics, "Moment of Inertia: Cylinder," [Online]. Available: <http://hyperphysics.phy-astr.gsu.edu/hbase/icyl.html>. [Accessed 2017 April 22].
- [41] S. Durovic, Mechatronic Analysis and Design, Manchester: The University of Manchester, 2016.

9. Appendices

9.1. Appendix 1 – MATLAB code

```
%robot parameters
mb = 0.987; %mass of robot
mw = 0.025; %mass of wheels
jb = 0.00383; %moment of inertia about the centre of mass
r = 0.04; %radius of wheels
jw = 4E-05; %moment of inertia for the wheels
l = 0.102; %distance from wheel axle to CoM
ke = 0.855;
km = 0.316;
R = 7.2; %motor resistance
b = 0.002; %Viscous friction constant
g = 9.81; %gravity

alp = (2*(R*b - ke*km)*(mb*l*l + mb*r*l + jb))/ R*(2*(jb*jw + jw*l*l*mb +
jb*mw*r*r + l*l*mb*mw*r*r)+jb*mb*r*r);

bet = (-l*l*mb*mb*g*r*r)/(jb*(2*jw + mb*r*r + 2*mw*r*r) + 2*jw*l*l*mb +
2*l*l*mb*mw*r*r);

gam = (-2*(R*b - ke*km)*(2*jw + mb*r*r + 2*mw*r*r + l*mb*r))/(R*r*(2*(jb*jw
+ jw*l*l*mb + jb*mw*r*r + l*l*mb*mw*r*r)+jb*mb*r*r));

delt = (l*mb*g*(2*jw + mb*r*r + 2*mw*r*r))/(2*jb*jw + 2*jw*l*l*mb +
jb*mb*r*r + 2*jb*mw*r*r + 2*l*l*mb*mw*r*r);

chi = (km*r)/(R*b - ke*km);

A = [0 1 0 0;
0 alp bet -r*alp;
0 0 0 1;
0 gam delt -r*alp];

B = [0; alp*chi; 0; gam*chi];

C = [1 0 0 0;
0 1 0 0;
0 0 1 0;
0 0 0 1]
D = [0;0;0;0]

Q=C'*C

[n,d]=ss2tf(A,B,C,D)

G = ss(A,B,C,D)

R = 1;

[K,S,e] = lqr(G,Q,R)
```

9.2. Appendix 2 – Arduino PID code

```
//Sources used in code development
//http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-
to-implement-it/
//http://playground.arduino.cc/Main/RotaryEncoders
//http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-
complementary-filter/
//http://www.x-firm.com/?page_id=191
//http://playground.arduino.cc/Code/PwmFrequency
// http://tom.pycke.be/mav/71/kalman-filtering-of-imu-data

#include <Wire.h>
#define Q_angle 0.03
#define Q_gyro 0.01
#define R 0.05

#define EN34 9 //m2 enable
#define EN12 10 //m1 enable
#define M2neg 6
#define M2pos 7
#define M1neg 5
#define M1pos 4

float summation=0;
float kp=100.0, ki=0.0, kd=0.0, output=0;
char offset = 0;

double gyroX, gyroY, gyroZ;//raw values
long accelX, accelY, accelZ;
float acc_x_zero, acc_y_zero, acc_z_zero, gyro_x_zero;

float g_x, g_y, g_z; //scalled values
float ang_vel_x, ang_vel_y, ang_vel_z;

float P_00 = 5, P_01 = 0, P_10 = 0, P_11 = 5;
double tilt_angle;
```

```

float bias_x;
float angle;
uint32_t timer;
uint32_t Test_timer;

void setPwmFrequency(int pin, int divisor) {
    byte mode;
    if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
        switch(divisor) {
            case 1: mode = 0x01; break;
            case 8: mode = 0x02; break;
            case 64: mode = 0x03; break;
            case 256: mode = 0x04; break;
            case 1024: mode = 0x05; break;
            default: return;
        }
        if(pin == 5 || pin == 6) {
            TCCR0B = TCCR0B & 0b11111000 | mode;
        } else {
            TCCR1B = TCCR1B & 0b11111000 | mode;
        }
    } else if(pin == 3 || pin == 11) {
        switch(divisor) {
            case 1: mode = 0x01; break;
            case 8: mode = 0x02; break;
            case 32: mode = 0x03; break;
            case 64: mode = 0x04; break;
            case 128: mode = 0x05; break;
            case 256: mode = 0x06; break;
            case 1024: mode = 0x07; break;
            default: return;
        }
        TCCR2B = TCCR2B & 0b11111000 | mode;
    }
}

void zero_calculation(){

```

```

//Serial.println("Begin Bias Measurement");
for(int i =0; i<50; i++){
    ReadData();
    gyro_x_zero= gyro_x_zero+ ang_vel_x;

    acc_x_zero = acc_x_zero + g_x;
    acc_y_zero = acc_y_zero + g_y;
    acc_z_zero = acc_z_zero + g_z;

    delayMicroseconds(2500);
}
gyro_x_zero = gyro_x_zero/50;

acc_x_zero = acc_x_zero/50;
acc_y_zero = acc_y_zero/50;
acc_z_zero = (acc_z_zero/50) - 1;

bias_x = gyro_x_zero; // initializing the bias

//Serial.println("Sequence Completed");
}
void KF(float newAngle, float newRate, float dt) {
    tilt_angle += dt * (newRate - bias_x);

    P_00 += dt * (dt*P_11 - P_01 - P_10 + Q_angle);
    P_01 -= dt * P_11;
    P_10 -= dt * P_11;
    P_11 += Q_gyro * dt;

    float S = P_00 + R;

    float K_0 = P_00 / S;
    float K_1 = P_10 / S;

    float y = newAngle - tilt_angle;

```

```

    tilt_angle += K_0 * y;
    bias_x += K_1 * y;

    float P00_temp = P_00;
    float P01_temp = P_01;

    P_00 -= K_0 * P00_temp;
    P_01 -= K_0 * P01_temp;
    P_10 -= K_1 * P00_temp;
    P_11 -= K_1 * P01_temp;
};

void motorctrl(int torque){    //torque between 0-255
    if (torque >= 0) {        // drive motors forward
        digitalWrite(M1neg, LOW);
        digitalWrite(M1pos, HIGH);
        digitalWrite(M2neg, HIGH);
        digitalWrite(M2pos, LOW);
        torque = abs(torque)+ offset;
    }
    else{                    // drive motors backward
        digitalWrite(M1neg, HIGH);
        digitalWrite(M1pos, LOW);
        digitalWrite(M2neg, LOW);
        digitalWrite(M2pos, HIGH);
        torque = abs(torque)+ offset;
    }

    analogWrite(EN12,torque);
    analogWrite(EN34,torque);
}

void MPUsetup(){
    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x19); // Set Sample rate to 1000Hz
    Wire.write(0x00); //
    Wire.endTransmission();
}

```

```
Wire.beginTransmission(0x1A); //I2C address of the MPU (as SJ2 is in place)
Wire.write(0x00); // Disable FSYNC
Wire.write(0x00); //
Wire.endTransmission();
```

//SETTING UP POWER

```
Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
Wire.write(0x6B); // Power Management 1
Wire.write(0x00); // pg 40
Wire.endTransmission();
```

```
Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
Wire.write(0x6C); // Power Management 2
Wire.write(0x00); // pg 41 - enables gyro and acc x,y,z
Wire.endTransmission();
```

//GYRO CONFIGURATION

```
Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
Wire.write(0x1B); // gyro configuration
Wire.write(0x02); // pg 14 - sets the full scale to +/- 1000 degress/second
Wire.endTransmission();
```

//ACC CONFIGURATION

```
Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
Wire.write(0x1C); // acc configuration
Wire.write(0x00); // pg 14 - sets the full scale to +/- 2gs
Wire.endTransmission();
```

```
}
```

```
void ReadData(){
```

```
    //get raw data (does not represent gs or dps, needs to be scalled depending on
    setup)
```

```
//accelerometer readings
```

```
Wire.beginTransmission(0x68); //I2C address of the MPU
```

```

Wire.write(0x3B); //Starting register for Accel Readings
Wire.endTransmission();
Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
while(Wire.available() < 6);
accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ

//gyro data

Wire.beginTransmission(0x68); //I2C address of the MPU
Wire.write(0x43); //Starting register for Gyro Readings
Wire.endTransmission();
Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
while(Wire.available() < 6);
gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into gyroX
gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into gyroY
gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into gyroZ

// scaling based on the set up full range
g_x = accelX / 16384.0;
g_y = accelY / 16384.0;
g_z = accelZ / 16384.0;

ang_vel_x = gyroX / (131.0);
ang_vel_y = gyroY / (131.0);
ang_vel_z = gyroZ / (131.0);
}

void setup() {
  Serial.begin(115200);
  Wire.begin();
  Wire.setClock(400000UL); // Set I2C frequency to 400kHz
  MPUsetup();
  delay(100); // Wait for sensor to stabilize
  setPwmFrequency(10, 1);
  pinMode(EN34, OUTPUT);
}

```



```

pinMode(EN12, OUTPUT);
pinMode(M2neg, OUTPUT);
pinMode(M2pos, OUTPUT);
pinMode(M1neg, OUTPUT);
pinMode(M1pos, OUTPUT);

ReadData();
zero_calculation(); //estimation of bias

float acc_xangle = atan2(g_y - acc_y_zero, g_z - acc_z_zero) * 57.3;
tilt_angle = acc_xangle; // Set starting angle
}
void loop() {
  ReadData();
  float dt = micros()- Test_timer;
  Test_timer = micros();

  dt /= 1000000;

  float acc_xangle = atan(g_y /g_z) * 57.3;

  KF(acc_xangle, ang_vel_x, dt);

  Serial.println(tilt_angle);

  kp = analogRead(A0)/4.0;
  kd= analogRead(A1)/128.0;
  ki= analogRead(A2)/128.0;

  summation = constrain( summation + tilt_angle*dt, 40, 40) ;
  int tiltoutput =constrain( kp*(tilt_angle) + kd*(ang_vel_x-bias_x) + ki*summation, -
254+offset, 254-offset) ;
  motorctrl(tiltoutput);
  while(micros() - timer < 5000); //200Hz
  timer = micros(); }

```

9.3. Appendix 3 – Arduino Motor Characterisation Code

```
#include <Wire.h>

#define EN34 9 //m2 enable
#define EN12 10 //m1 enable
#define M2neg 6
#define M2pos 7
#define M1neg 5
#define M1pos 4
int offset =10;
int encoder0PinA = 2;
int encoder0PinB = 12;
volatile long counter=0;
double outer;
int prev_counter;

void setPwmFrequency(int pin, int divisor) {
  byte mode;
  if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
      case 64: mode = 0x03; break;
      case 256: mode = 0x04; break;
      case 1024: mode = 0x05; break;
      default: return;
    }
    if(pin == 5 || pin == 6) {
      TCCR0B = TCCR0B & 0b11111000 | mode;
    } else {
      TCCR1B = TCCR1B & 0b11111000 | mode;
    }
  } else if(pin == 3 || pin == 11) {
    switch(divisor) {
      case 1: mode = 0x01; break;
      case 8: mode = 0x02; break;
```

```

        case 32: mode = 0x03; break;
        case 64: mode = 0x04; break;
        case 128: mode = 0x05; break;
        case 256: mode = 0x06; break;
        case 1024: mode = 0x07; break;
        default: return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
}
}

void motorctrl(int torque){    //torque between 0-255
    if (torque >= 0) {        // drive motors forward
        digitalWrite(M1neg, LOW);
        digitalWrite(M1pos, HIGH);
        digitalWrite(M2neg, HIGH);
        digitalWrite(M2pos, LOW);
        torque = abs(torque)+ offset;
    }
    else{                    // drive motors backward
        digitalWrite(M1neg, HIGH);
        digitalWrite(M1pos, LOW);
        digitalWrite(M2neg, LOW);
        digitalWrite(M2pos, HIGH);
        torque = abs(torque)+ offset;
    }

    analogWrite(EN12,torque);
    analogWrite(EN34,torque);
}

void setup() {
    attachInterrupt(digitalPinToInterrupt(encoder0PinA), rotary, RISING);
    pinMode (encoder0PinB,INPUT);

    Serial.begin(115200);

```

```

Wire.begin();
Wire.setClock(400000UL); // Set I2C frequency to 400kHz
delay(100); // Wait for sensor to stabilize

setPwmFrequency(10, 1);

pinMode(EN34, OUTPUT);
pinMode(EN12, OUTPUT);
pinMode(M2neg, OUTPUT);
pinMode(M2pos, OUTPUT);
pinMode(M1neg, OUTPUT);
pinMode(M1pos, OUTPUT);
}
void loop() {

  for (int x = 0; x<52; x++){
    counter =0;
    motorctrl(x*5);
    delay(5000);
    motorctrl(0*5);
    float rads = counter*((2*3.14159)/(562*5));
    Serial.print(x*5);
    Serial.print("\t");
    Serial.println(rads);
  }

}

void rotary() {
// Serial.println(counter);
  if(digitalRead(encoder0PinB)) {
    counter++;
  } else {
    counter--;
  }
}
}

```

9.4. Appendix 4 – Arduino LQR Code

[//http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/](http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/)

[//http://playground.arduino.cc/Main/RotaryEncoders](http://playground.arduino.cc/Main/RotaryEncoders)

[//http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/](http://www.geekmomprojects.com/mpu-6050-redux-dmp-data-fusion-vs-complementary-filter/)

[//http://www.x-firm.com/?page_id=191](http://www.x-firm.com/?page_id=191)

```
#include <Wire.h>
```

```
#define Q_angle 0.03
```

```
#define Q_gyro 0.01
```

```
#define R 0.05
```

```
#define Pi 3.14159
```

```
#define EN34 9 //m2 enable
```

```
#define EN12 10 //m1 enable
```

```
#define M2neg 6
```

```
#define M2pos 7
```

```
#define M1neg 5
```

```
#define M1pos 4
```

```
int encoder0PinA = 2;
```

```
int encoder0PinB = 12;
```

```
volatile long counter=0;
```

```
uint32_t previous_counter=0;
```

```
int loop_iteration =0;
```

```
char offset = 0;
```

```
float K1 = 1.0000, K2=0.1129, K3=-37.7177, K4=-1.4959;
```

```
float x1, x2, x3, x4;
```

```
double gyroX, gyroY, gyroZ;//raw values
```

```
long accelX, accelY, accelZ;
```

```
float acc_x_zero, acc_y_zero, acc_z_zero, gyro_x_zero;
```

```
float g_x, g_y, g_z; //scaled values
```

```
float ang_vel_x, ang_vel_y, ang_vel_z;
```

```
float P_00 = 5, P_01 = 0, P_10 = 0, P_11 = 5;
```

```
double tilt_angle;
```

```
float bias_x;
```

```
float angle;
```

```
uint32_t timer;
```

```
uint32_t Test_timer;
```

```
void setPwmFrequency(int pin, int divisor) {
```

```
    byte mode;
```

```
    if(pin == 5 || pin == 6 || pin == 9 || pin == 10) {
```

```
        switch(divisor) {
```

```
            case 1: mode = 0x01; break;
```

```
            case 8: mode = 0x02; break;
```

```
            case 64: mode = 0x03; break;
```

```
            case 256: mode = 0x04; break;
```

```
            case 1024: mode = 0x05; break;
```

```
            default: return;
```

```
        }
```

```
        if(pin == 5 || pin == 6) {
```

```
            TCCR0B = TCCR0B & 0b11111000 | mode;
```

```
        } else {
```

```
            TCCR1B = TCCR1B & 0b11111000 | mode;
```

```
        }
```

```
    } else if(pin == 3 || pin == 11) {
```

```
        switch(divisor) {
```

```
            case 1: mode = 0x01; break;
```

```
            case 8: mode = 0x02; break;
```

```
            case 32: mode = 0x03; break;
```

```
            case 64: mode = 0x04; break;
```

```
            case 128: mode = 0x05; break;
```

```
            case 256: mode = 0x06; break;
```

```

        case 1024: mode = 0x07; break;
        default: return;
    }
    TCCR2B = TCCR2B & 0b11111000 | mode;
}
}

void zero_calculation(){
//Serial.println("Begin Bias Measurement");
for(int i =0; i<50; i++){
    ReadData();
    gyro_x_zero= gyro_x_zero+ ang_vel_x;

    acc_x_zero = acc_x_zero + g_x;
    acc_y_zero = acc_y_zero + g_y;
    acc_z_zero = acc_z_zero + g_z;

    delayMicroseconds(2500);
}
gyro_x_zero = gyro_x_zero/50;

acc_x_zero = acc_x_zero/50;
acc_y_zero = acc_y_zero/50;
acc_z_zero = (acc_z_zero/50) - 1;

bias_x = gyro_x_zero; // initializing the bias

//Serial.println("Sequence Completed");
}

void KF(float newAngle, float newRate, float dt) {
    tilt_angle += dt * (newRate - bias_x);

    P_00 += dt * (dt*P_11 - P_01 - P_10 + Q_angle);
    P_01 -= dt * P_11;
    P_10 -= dt * P_11;
    P_11 += Q_gyro * dt;

```

```

float S = P_00 + R;

float K_0 = P_00 / S;
float K_1 = P_10 / S;

float y = newAngle - tilt_angle;

tilt_angle += K_0 * y;
bias_x += K_1 * y;

float P00_temp = P_00;
float P01_temp = P_01;

P_00 -= K_0 * P00_temp;
P_01 -= K_0 * P01_temp;
P_10 -= K_1 * P00_temp;
P_11 -= K_1 * P01_temp;
};

void motorctrl(int torque){    //torque between 0-255
    if (torque >= 0) {        // drive motors forward
        digitalWrite(M1neg, LOW);
        digitalWrite(M1pos, HIGH);
        digitalWrite(M2neg, LOW);
        digitalWrite(M2pos, HIGH);
        torque = abs(torque)+ offset;
    }
    else{                    // drive motors backward
        digitalWrite(M1neg, HIGH);
        digitalWrite(M1pos, LOW);
        digitalWrite(M2neg, HIGH);
        digitalWrite(M2pos, LOW);
        torque = abs(torque)+ offset;
    }

    analogWrite(EN12,torque);

```



```

    analogWrite(EN34,torque);
}

void MPUsetup(){
    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x19); // Set Sample rate to 1000Hz
    Wire.write(0x00); //
    Wire.endTransmission();

    Wire.beginTransmission(0x1A); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x00); // Disable FSYNC
    Wire.write(0x00); //
    Wire.endTransmission();

    //SETTING UP POWER
    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x6B); // Power Management 1
    Wire.write(0x00); // pg 40
    Wire.endTransmission();

    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x6C); // Power Management 2
    Wire.write(0x00); // pg 41 - enables gyro and acc x,y,z
    Wire.endTransmission();

    //GYRO CONFIGURATION
    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x1B); // gyro configuration
    Wire.write(0x02); // pg 14 - sets the full scale to +/- 1000 degress/second
    Wire.endTransmission();

    //ACC CONFIGURATION
    Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
    Wire.write(0x1C); // acc configuration
    Wire.write(0x00); // pg 14 - sets the full scale to +/- 2gs
    Wire.endTransmission();

```

```

}
void ReadData(){
    //get raw data (does not represent gs or dps, needs to be scaled depending on
    setup)

    //accelerometer readings
    Wire.beginTransmission(0x68); //I2C address of the MPU
    Wire.write(0x3B); //Starting register for Accel Readings
    Wire.endTransmission();
    Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
    while(Wire.available() < 6);
    accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
    accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
    accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ

    //gyro data

    Wire.beginTransmission(0x68); //I2C address of the MPU
    Wire.write(0x43); //Starting register for Gyro Readings
    Wire.endTransmission();
    Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
    while(Wire.available() < 6);
    gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into gyroX
    gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into gyroY
    gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into gyroZ

    // scaling based on the set up full range
    g_x = accelX / 16384.0;
    g_y = accelY / 16384.0;
    g_z = accelZ / 16384.0;

    ang_vel_x = gyroX / (32.768);
    ang_vel_y = gyroY / (32.768);
    ang_vel_z = gyroZ / (32.768);
}
void setup() {

```

```

attachInterrupt(digitalPinToInterrupt(encoder0PinA), rotary, RISING);
pinMode (encoder0PinB,INPUT);

Serial.begin(115200);
Wire.begin();
Wire.setClock(400000UL); // Set I2C frequency to 400kHz
MPUsetup();
delay(100); // Wait for sensor to stabilize
setPwmFrequency(10, 1);
pinMode(EN34, OUTPUT);
pinMode(EN12, OUTPUT);
pinMode(M2neg, OUTPUT);
pinMode(M2pos, OUTPUT);
pinMode(M1neg, OUTPUT);
pinMode(M1pos, OUTPUT);

ReadData();
zero_calculation(); //estimation of bias

float acc_xangle = atan2(g_y - acc_y_zero, g_z - acc_z_zero) * 57.3;
tilt_angle = acc_xangle; // Set starting angle

timer = micros();
}
void rotary() {
// Serial.println(counter);
if(digitalRead(encoder0PinB)) {
    counter++;
} else {
    counter--;
}
}
void loop() {
    ReadData();
    float dt = micros()- Test_timer;

```

```

Test_timer = micros();
dt /= 1000000;

float acc_xangle = atan(g_y /g_z) * 57.3;
KF(acc_xangle, ang_vel_x, dt);

if (loop_ititeration==10){
x2 = ((counter-previous_counter)*2*Pi)/0.05;//linear velocity
counter=previous_counter;
loop_ititeration=0;}
loop_ititeration++;

x1 = (counter*2*Pi*0.04)/562; //displacement
x3 = (tilt_angle)/57.3; //tilt angle in radians
x4 = (ang_vel_x - bias_x)/0.0174533; //angular velocity in rad/s

float tiltoutput = constrain(-(x1*K1+x2*K2+x3*K3+x4*K4)*(255/12), -254, 254);
motorctrl(tiltoutput);

while(micros() - timer < 5000); //200Hz
timer = micros();
}

```

9.5. Appendix 5 – Progress Report



The University of Manchester

Self-Balancing Robot

Third Year Individual Project – Progress Report

Nov 2016

Abdul Gafar

9097951

Supervisor:

Dr. Joaquin Carrasco Gomez

Contents

1. Introduction and Motivation	1
2. Aims and Objectives	2
3. Existing Work	2
4. Kalman Filter	3
4.1. Creating a Model	3
4.2. The Kalman Filter Algorithm	4
4.2.1. Time Update	4
4.2.2. Measurement Update	5
4.3. Overall Diagram	5
4.4. Kalman Filter Practice in MATLAB	5
5. Hardware	7
5.1. Microcontroller	7
5.2. Motors	7
5.3. Power Source	8
5.4. Motor Driver Board	8
5.5. IMU	8
5.6. Overall Design	9
6. Conclusion	10
7. References	11
8. Appendices	13
8.1. Appendix 1 –Technical Risk Assessment	13
8.2. Appendix 2 – Health and Safety Risk Assessment	14
8.3. Appendix 3– Project Plan	16
8.4. Appendix 4 -Kalman Filter Code 1 – Constant	17
8.5. Appendix 5 - Kalman Filter Code 2 – Linear	18
8.6. Appendix 6 - MPU 9250 Register Map	19
8.7. Appendix 7 - IMU Code to obtain raw values	22
8.8. Appendix 8 – IMU Output	24

1. Introduction and Motivation

Self-balancing robots have sparked interest of many researchers, students and hobbyist worldwide. From an engineer's perspective, it is an inverted pendulum on wheels. The inverted pendulum is a classical problem in control systems due its unstable nature. To the average individual, one of the triggers for the curiosity towards the self-balancing robots was the release of the Segway PT (Personal Transporter). These robots became very popular because of their manoeuvrability, in particular their short turning radius [1]. The Segway has been used in many industries, from tourism in the park, police, and even ambulances. In recent times, a derivative of the Segway, the hoverboard, has been a headline in social media, once again directing the attention of many towards the engineering behind.

In any balancing robot knowing the tilt angle is critical, thus an inertial measurement unit (IMU) is a necessity. The IMU is predominantly composed of a gyroscope and an accelerometer. Both sensors have their advantages and disadvantages, therefore to obtain a more accurate measurement the data has to be fused. As part of the project, a technique known as Kalman filtering will be explored. If implemented and tuned correctly, the Kalman Filter best possible (optimal) estimator for a large class of problems." [2]

As a Mechatronics student, making a self-balancing robot is the ideal project. The core of the project is control, thus it will allow the application what has been covered to date and exploration of new material such as alternative controllers, data fusion or odometry. In addition, the project is sufficiently broad to refine knowledge in the areas of embedded systems, programming, PCB and mechanical design. The material to be covered has a broad range of applications, developing many skills transferrable to future projects.

The purpose of this report is to outline the plan of the project and to summarize the progress achieved to date.

2. Aims and Objectives

The aim of the project is to design, make and program a Self-Balancing Robot with a self-developed Kalman Filter. In order to successfully complete the project, the following objectives need to be met:

- Perform Literature review on Kalman Filters and implement in MATLAB
- Develop a Kalman Filter to fuse data from the gyroscope and accelerometer
- Design and assemble the chassis of the robot
- Develop a PID controller to enable the robot to stay upright

If time permits, the list below outlines the possible additional targets:

- Explore the use of a LQR or Fuzzy Logic controller
- Create a remote controller for the robot
- Improve the control algorithm to be able to support loads including asymmetrical loads
- Create Autonomous Pre-programmed paths using odometry

3. Existing Work

Balancing Robots have existed for several years, thus many papers and theses have been written about them. Some are purely for learning purposes, as is the case. Others are to research the application of certain theory such as the LQR controller or fuzzy logic. And in certain theses, it is develop a robot for a specific purpose, this includes a butler robot or an interactive balancing robot to be used in exhibitions.

In most cases, students would focus on a certain aspect, such as data fusion, analysis of dynamics or controller design, and the rest of the robot would be built using simpler techniques. For example, they would focus on using a Kalman filter and use a PID controller or focus on LQR controller and use a Complementary filter.

For sensor fusion, the complementary filter and the Kalman filter are the most commonly techniques. The Kalman filter will be further explained in section 4. The complementary filter, simply consists of a low pass filter for the gyroscope and high pass filter for the accelerometer. Whilst, the Kalman filter is accepted as the best estimator, in a specific case the complementary filter appeared to perform better. [3]

To maintain the robot upright, the commonly mentioned controllers are Proportional-

integral-derivative (PID) and the Linear Quadratic Regulator (LQR). A Linear Quadratic-Gaussian controller has also been tested, however, due to a slow microcontroller, it was not successful. [1] In more complex balancing robots, which the robot also moves around, two controllers are used. For example an LQR controller to balance the robot and a PID controller to control yaw. [4]

4. Kalman Filter

The Kalman Filter (KF) was first introduced in 1960 by Rudolf E. Kalman [5]. Since then, due to its adaptability and usefulness, research and development has continued creating variants such as the Extended Kalman Filter or the Unscented Kalman Filter [2]. The KF was famously used in the Apollo program, ultimately taking Neil Armstrong to the moon [6]. “The Kalman Filter is over 50 years old but is still one of the most important data fusion algorithms in use today [7].” Its use ranges from navigation and object tracking to investment banking and economics.

Data fusion is essential in this case due to the nature of the gyroscope and accelerometer. The accelerometer measurements are more susceptible to noise, whilst the gyroscope drifts over time. This makes the accelerometer readings more accurate in the long run, and the gyroscope more accurate over a short space of time [8]. To resolve the dilemma the KF can be used.

In addition to the accuracy of estimation, the KF is appealing because it is a recursive method. The current state is dependent on the previous state, which means that not all the data is necessary, allowing it to be implemented in a simple microcontroller without large storage [9]. One of the barriers for use of the KF is difficulty in understanding due to the lack of standard notation.

4.1. Creating a Model

To implement a KF, the system needs to be modelled in state-space form. The difference equation (1) that can be used to represent the process state and equation (2) models the measurements [2].

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

$$z_k = Hx_k + v_k \dots \dots \dots (2)$$

Where [6]:

x_k is the state vector, contains variables to be estimated i.e. angle or bias

u_k is the vector containing control inputs i.e. angular acceleration

A is the transition matrix, which maps the state parameters at $t-1$ to t

B is the control input matrix, maps the controlled inputs u_k to the state vector

z_k is the measurements matrix

H is matrix that transforms the state vector into measurements

w_k and v_k are the vectors containing the process noise and measurement noise respectively. The noise is assumed to be zero mean Gaussian distributed with a covariance Q and R, respectively i.e. $w_k \sim (0, Q)$ and $v_k \sim (0, R)$.

4.2. The Kalman Filter Algorithm

The KF is composed of two sets of equations, time update and measurement update equations.

4.2.1. Time Update

The following equations describe the time update stage, also known as prediction stage:

$$\hat{x}_{k|k-1} = A\hat{x}_{k-1|k-1} + Bu_k \dots \dots \dots (3)$$

$$P_{k|k-1} = A_k P_{k-1|k-1} A_k^T + Q_k \dots \dots \dots (4)$$

Where:

\hat{x} is the state estimate

P is the process covariance matrix

A note on the subscript: **a | b** would mean **a** given **b** and all previous states before **b**. For example $\hat{x}_{k|k-1}$, is the estimate at k based on k-1 and on all the states before k-1. $\hat{x}_{k|k-1}$ is known as the priori state, $\hat{x}_{k-1|k-1}$ is the previous state and $\hat{x}_{k|k}$ is the posteriori state.

4.2.2. Measurement Update

The following equations are used in the measurement update:

$$K_k = P_{k|k-1} H_k^T (H_k P_{k|k-1} H_k^T + R_k)^{-1} \dots\dots\dots (5)$$

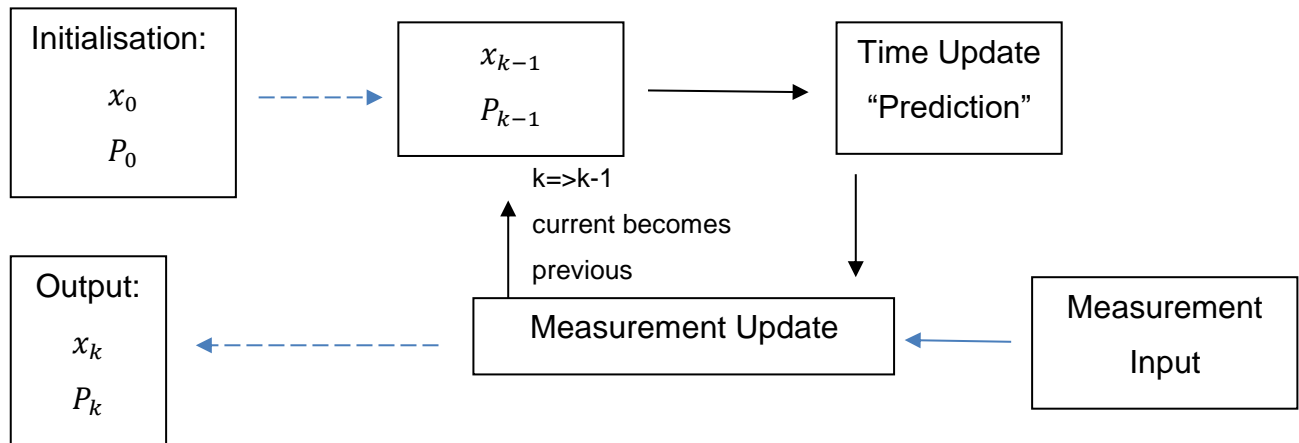
$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k (z_k - H \hat{x}_{k|k-1}) \dots\dots\dots (6)$$

$$P_{k|k} = P_{k|k-1} - K_k H_k P_{k|k-1} \dots\dots\dots (7)$$

Where: K is the Kalman Gain Matrix

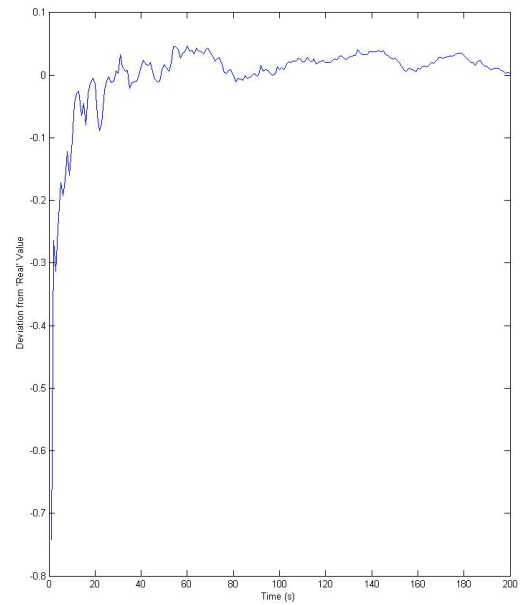
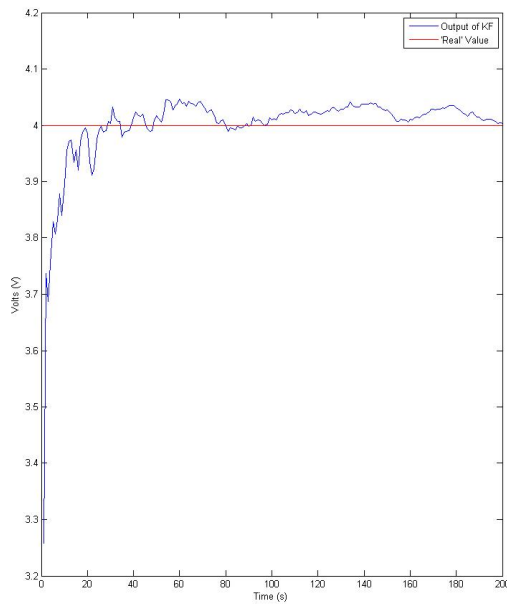
4.3. Overall Diagram

The KF runs in a loop shown in the diagram below:

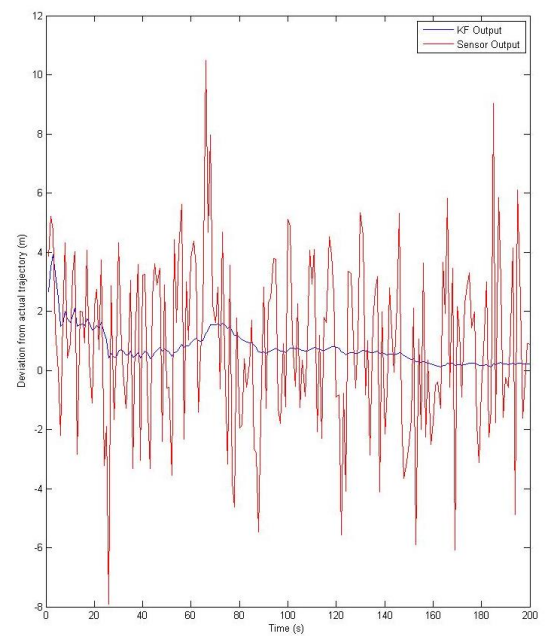
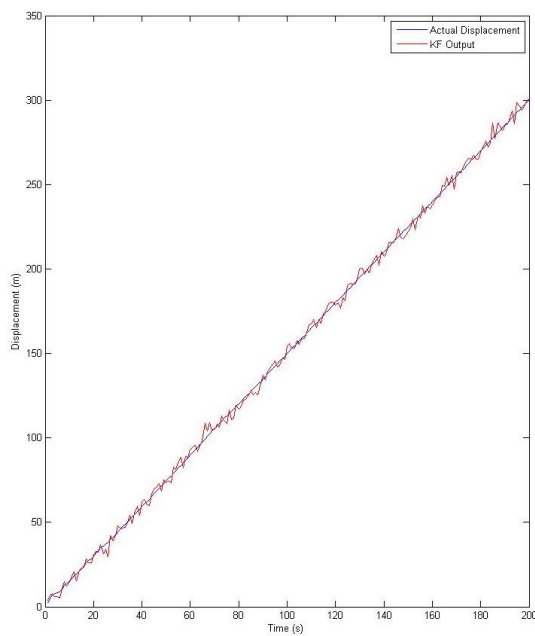


4.4. Kalman Filter Practice in MATLAB

In order to better understand how KFs are implemented, examples were done in MATLAB. The first example was following a tutorial, which the ‘real’ measurement was a constant voltage [10]. In the tutorial the computation was shown, but no code was given. Implementing it in MATLAB helped visualize how the KF can be realised in code. The MATLAB code can be found in **Appendix B**. The figure in the following shows the output:



To further aid understanding, a simple example was created and implemented. It consists of measuring the displacement of an object travelling in 1-D at a constant velocity of 1.5m/s. The MATLAB code can be found in **Appendix C**. The figure below shows the output:



5. Hardware

5.1. Microcontroller

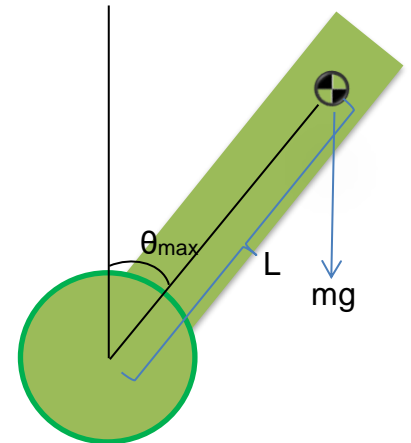
The microcontroller chosen was the Arduino Uno. It has a relatively small footprint, keeping the robot compact. The main advantage of the Arduino is large community and extensive collection of libraries, if any problems are stumbled upon, there is a higher chance that someone else has found a solution.

5.2. Motors

In order to establish the motors required, a calculation of the required torque is necessary. The diagram to the right shows a sketch of the balancing robot.

$$\tau = \|\mathbf{r}\| \|\mathbf{F}\| \sin\theta \dots\dots\dots (1)$$

Where: τ is magnitude of the torque, \mathbf{F} is the force vector, \mathbf{r} is the position vector and θ is the angle between force and position vectors.



Assuming the distance between the pivot point and the centre of mass (L) is 12cm, the maximum tilt angle (θ_{\max}) is 40° and the mass of the robot (m) is 0.7kg.

$$\tau = L * mg * \sin\theta = 0.12 * 0.7 * 9.81 * \sin(40) = 0.530 \text{ Nm} \dots\dots\dots (2)$$

Since there will be two motors, the minimum torque required is 0.265Nm. This assumes the robot is going to start moving at the maximum tilt angle, in reality inertia also has to be considered.

Looking at practical example, Gornicki used motors with a stall torque of 0.224Nm and a gear with a 3:1 ratio [11]. Assuming 15% inefficiency [12], that equates to 0.5712Nm.

To fit the requirements, the chosen motor is the Pololu medium power 47:1 Metal Gearmotor with 48 CPR Encoder. The stall torque of the motor is 0.611Nm and the encoder outputs 2248.86 counts per revolution [13], corresponding to a resolution of up to 0.16° . The encoders are necessary for odometry, without the encoders the robot may balance but it will be moving around constantly.

5.3. Power Source

The considered power sources were lithium polymer (Li-Po) batteries and AA batteries. Li-Po batteries were found the most appropriate power source, as AA batteries generally have a lower maximum discharge current [14]. Li-Po batteries also have a relatively high specific energy and energy density [15]. There are some dangers associated with them, these have been addressed in the Health and Safety Risk Assessment (**Appendix x**). The specific battery to be used is the Turnigy 3 cell 2200mAh 20C. The stall current for each motor is 2.1A at 12V [13] and power also needs to be supplied to the other devices (Arduino, IMU and encoders). As a rough estimate, the power source should be able to supply a minimum of 5A. The Li-Po battery can supply up to 44A [16].

5.4. Motor Driver Board

The L298 dual full bridge driver was initial choice. According to the datasheet the motor driver has peak output current per channel of 2A in DC operation and up to 3A non-repetitive [17]. In practice, the L298 would go into thermal shut down at 0.8A [18], making it unsuitable for the robot. To avoid deceit from manufacturers, the L6203 was chosen, theoretically it can supply 5A [19]. In order not to damage the motors resettable fuses will be used.

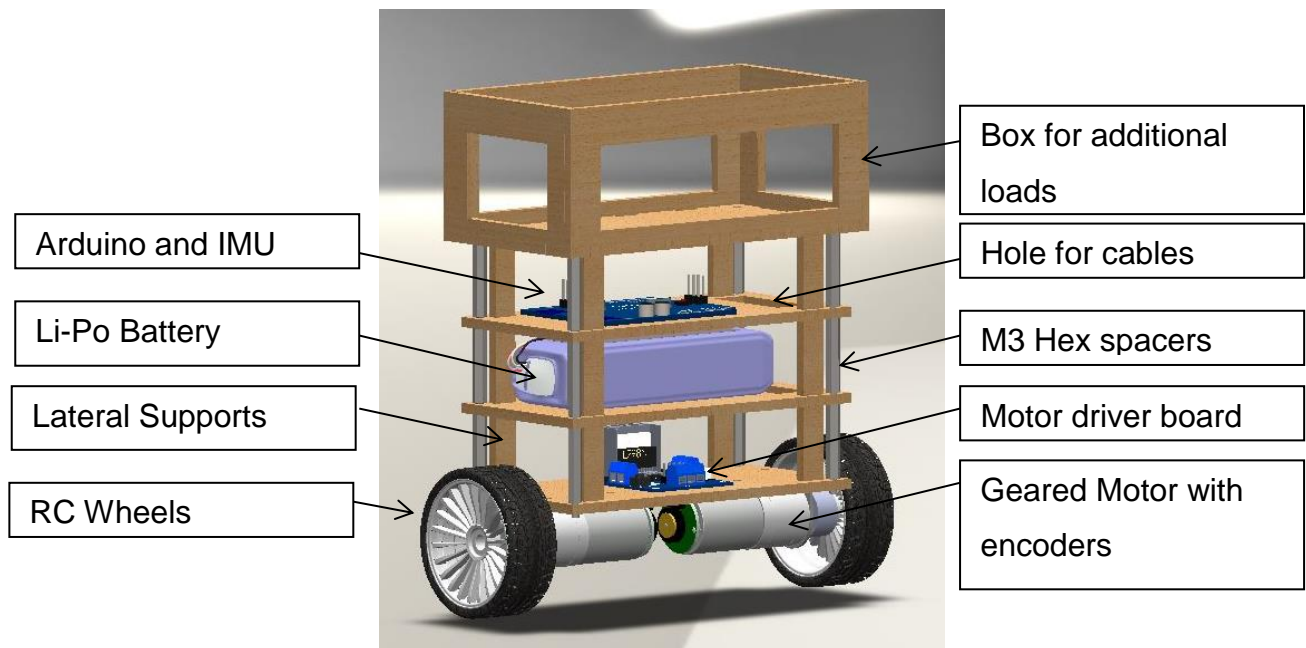
5.5. IMU

The selected IMU is the MPU 9250 by InvenSense. It has 9 degrees of freedom, consisting of 3-axis gyroscope, 3-axis accelerometer and 3-axis magnetometer. The magnetometer is not necessary, but the IMU without the magnetometer costs twice the price. By accessing the configuration register, the gyroscope full scale range can be adjusted from ± 250 to 1000 degrees per second. The accelerometer can also be programmed from ± 2 to 6 g. This device has a built in Digital Motion Processor (DMP), but for this project it will not be used. A great advantage of this IMU is that it has been used with the Arduino and libraries are available for it. [20]

Communication between the Arduino and the IMU is through the Inter-Integrated Circuit (I2C) protocol. To read the values from the gyroscope and accelerometer, specific memory addresses need to be accessed (the register map is in the **appendix**). Following a tutorial for the MPU6050, the raw data values were read. Surprisingly, the register map for MPU9250 is identical to the MPU6050. The code to read the values and the output window is in the **Appendix X**.

5.6. Overall Design

The overall planned format of the robot can be seen in the Solidworks render below:



The design is an adaptation of the SainSmart self-balancing robot [21]. The design is entirely modular. The layer heights can be changed by changing the spacer lengths and the box for loads can be removed. Having the layers also protects the components, specifically the Li-Po battery. The battery is shielded from heat from the motor drivers and it is also protected from impacts.

The layers will be made of Medium Density Fibreboard (MDF). It is relatively light, inexpensive, easy to manufacture and readily available in the university. In addition MDF should be able to withstand the drops and hits that might happen when the robot controller is being tuned.

The wheels will be from Remote Controlled (RC) cars. They are wide and the tyres made of soft rubber. This enables the wheels to have good grip, which is not surprising as often RC hobbyists compete with each other.

6. Conclusion

A basic understanding of Kalman filters has been achieved and the robot's physical design has been completed. The next step this semester is to implement the KF in C code to fuse the data from the gyroscope and accelerometer. A comparison can then be made between the data from the output of the KF and the built in DMP. Once the Kalman filter is well tuned and a good estimate of the tilt angle is obtained, the PID controller can then be developed to maintain the robot upright.

The progress achieved to date is as planned, this suggests that the aim of the project is realistic. Based on the Gantt chart in **Appendix A**, the project should be completed by the end of week 6 in second semester, allowing some time to adjust for unpredicted scenarios or to be dedicated in meeting the additional objectives.

7. References

- [1] Sundin, C. and Thorstensson, F. (2013). Autonomous balancing robot. Masters of Science. Chalmers University of Technology.
- [2] Welch, G. and Bishop, G. (2001). An Introduction to the Kalman Filter. [online] Available at:
http://www.cs.unc.edu/~tracker/media/pdf/SIGGRAPH2001_CoursePack_08.pdf [Accessed 4 Nov. 2016].
- [3] Bonafilia, B., Gustafsson, N., Nyman, P. and Nilsson, S. (n.d.). Self-balancing two-wheeled robot. Chalmers University of Technology.
- [4] Ding, Y., Gafford, J. and Kunio, M. (2012). Modeling, Simulation and Fabrication of a Balancing Robot. Harvard University, Massachusetts Institute of Technology.
- [5] Welch, G. and Bishop, G. (2006). An Introduction to the Kalman Filter. [online] Available at: http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf [Accessed 4 Nov. 2016].
- [6] Carrasco, J. (2016). 5 Lines of Code to Land on the Moon.
- [7] Faragher, R. (2012). Understanding the Basis of the Kalman Filter Via a Simple and Intuitive Derivation. IEEE SIGNAL PROCESSING MAGAZINE, [online] (1053-5888/12), pp.128-132. Available at:
<https://www.cl.cam.ac.uk/~rmf25/papers/Understanding%20the%20Basis%20of%20the%20Kalman%20Filter.pdf> [Accessed 7 Nov. 2016].
- [8] Cornman, A. and Mei, D. (n.d.). Extended Kalman Filtering. Stanford University.
- [9] Ooi, R. (2013). Balancing a Two-Wheeled Autonomous Robot. Undergraduate. The University of Western Australia.
- [10] Esme, B. (2016). Bilgin's Blog | Kalman Filter For Dummies. [online] [Bilgin.esme.org](http://bilgin.esme.org). Available at:
<http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies> [Accessed 4 Nov. 2016].
- [11] Gornicki, K. (2015). Autonomous Self Stabilising Robot. Undergraduate. The University of Manchester.
- [12] Baines, G. (2015) 'Embedded Systems Project: Motor Characterisation and Gearbox Ratio Selection'. Available at:
https://online.manchester.ac.uk/bbcswebdav/pid-3643166-dt-content-rid-12476223_1/courses/I3027-EEEN-21000-1151-1YR-

027927/ESP%20Week%202%20Motors%20and%20Gearbox_2015.pdf

(Accessed: 04/11/2015)

- [13] Pololu.com. (2016). Pololu - 47:1 Metal Gearmotor 25Dx52L mm MP 12V with 48 CPR Encoder. [online] Available at:
<https://www.pololu.com/product/3241/specs> [Accessed 3 Nov. 2016].
- [14] Energizer.com. (2016). Product Datasheet - L91 Ultimate Lithium. [online] Available at: <http://data.energizer.com/PDFs/l91.pdf> [Accessed 7 Nov. 2016].
- [15] Learn.sparkfun.com. (2016). Battery Technologies. [online] Available at:
<https://learn.sparkfun.com/tutorials/battery-technologies> [Accessed 4 Nov. 2016].
- [16] Hobbyking. (2016). Turnigy 2200mAh 3S 20C Lipo Pack. [online] Available at:
https://www.hobbyking.com/en_us/turnigy-2200mah-3s-20c-lipo-pack.html [Accessed 7 Nov. 2016].
- [17] Sparkfun. (2016). L298 H Bridge Datasheet. [online] Available at:
https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf [Accessed 7 Nov. 2016].
- [18] Rugged Circuits. (2016). The Motor Driver Myth. [online] Available at:
<http://www.rugged-circuits.com/the-motor-driver-myth/> [Accessed 3 Nov. 2016].
- [19] Anon, (2016). L6203 Datasheet. [online] Available at:
<http://users.ece.utexas.edu/~valvano/Datasheets/L6203.pdf> [Accessed 4 Nov. 2016].
- [20] InvenSense.com. (2016). MPU-9250 | InvenSense. [online] Available at:
<https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250> [Accessed 4 Nov. 2016].
- [21] Sainsmart.com. (2016). SainSmart 2-Wheel Arduino Self-Balancing Robot Kit 3D Printing, Arduino, Robotics | Sainsmart. [online] Available at:
<http://www.sainsmart.com/sainsmart-balancing-robot-kit.html> [Accessed 3 Nov. 2016].

8. Appendices

8.1. Appendix 1 –Technical Risk Assessment

As mentioned previously, the Arduino makes it an easy platform to program in due to large community and extensive collection of libraries. Furthermore, Kalman filters and balancing robots have been realized using an Arduino, this suggests a lower technical risk. However, due to low processing capability the Arduino itself may be a liability. Christian Sundin mentions that the Arduino could not execute the algorithm for an LQG controller fast enough [1]. If met with such scenario, a solution may be to use two Arduinos in master-slave configuration or a faster microcontroller such as the STM32 Nucleo.

Another risk for the project would be slow order processing time and delivery. If the required components do not arrive within the expected time frame, the project will have to be put on hold. To minimize this risk, component orders were placed early this semester.

8.2. Appendix 2 – Health and Safety Risk Assessment



The University of Manchester

WORK ACTIVITY/ WORKPLACE (WHAT PART OF THE ACTIVITY POSES RISK OF INJURY OR ILLNESS)	HAZARD (S) (SOMETHING THAT COULD CAUSE HARM, ILLNESS OR INJURY)	LIKELY CONSEQUENCES (WHAT WOULD BE THE RESULT OF THE HAZARD)	WHO OR WHAT IS AT RISK (INCLUDE NUMBERS AND GROUP S)	EXISTING CONTROL MEASURES IN USE (WHAT PROTECTS PEOPLE FROM THESE HAZARDS)	WITH EXISTING CONTROLS			WITH NEW CONTROLS		
					SEVERITY	LIKELIHOOD	RISK RATING	RISK RATING	LIKELIHOOD	RISK RATING
Soldering	High Temperature of the Soldering iron	Burns	Abdul Gafar and people nearby	Holder for iron, sponge to test if hot	2	2	4	Y	2	4
	Inhaling solder fumes	Illness		Fume extractor	3	1	3	Y	3	Y
Programming for the robot	Using a computer	RSI	Abdul Gafar	Ergonomic chairs	1	3	3	Y	1	3
		Eye strain		Limited computer usage	1	3	3	Y	1	3
Wiring the circuit	Using wire cutters	Cutting hands	Abdul Gafar	n/a	2	2	4	Y	2	4
		Small bits of wire flying off and hitting the eye			3	1	3	Y	3	Y
Using / Charging a Lithium Polymer Battery	Battery could explode if: overcharged, heated, the output terminals are shorted and/or is punctured	Damage equipment around and/or cause burns	Equipment in proximity and/or people nearby	Use a Balance charger, the battery will be shielded from heat or impact, a fuse should be used	4	2	8	Y	4	Y
	Damage the rest of the circuitry, due to high current	Some circuitry of the robot could be damaged	Robot gets damaged	Fuse	2	2	4	Y	2	4

Assessment ID Number (E&EE_AG_09/10/2016_9087951)..... Activity Location: SSB C34.....

THIS RISK ASSESSMENT WILL BE SUBJECT TO A REVIEW NO LATER THAN: (MAX 12 MTHS)	
MANAGER/SUPERVISOR	NAME: Dr. Joaquin Carrasco Gomez SIGNED: DATE:
Student:	NAME: Abdul Gafar SIGNED: DATE:

IF THE ANSWERS TO ANY OF THE QUESTIONS BELOW IS YES THEN ADDITIONAL SPECIFIC RISK ASSESSMENTS MAY BE REQUIRED.

IS THERE A RISK OF FIRE?	Y/N	DOES THE ACTIVITY REQUIRE ANY HOME WORKING?	Y/N
ARE SUBSTANCES THAT ARE HAZARDOUS TO HEALTH USED?	Y/N	ARE THE EMPLOYEES REQUIRED TO WORK ALONE	Y/N
IS THERE MANUAL HANDLING INVOLVED?	Y/N	DOES THE ACTIVITY INVOLVE DRIVING	Y/N
IS PPE WORN OR REQUIRED TO BE WORN?	Y/N	DOES THE ACTIVITY REQUIRE WORK AT HEIGHT	Y/N
ARE DISPLAY SCREENS USED?	Y/N	DOES THE ACTIVITY INVOLVE FOREIGN TRAVEL	Y/N
IS THERE A SIGNIFICANT RISK TO YOUNG PERSONS?	Y/N	IS THERE A SIGNIFICANT RISK TO NEW / PREGNANT MOTHERS?	Y/N

Severity value = potential consequence of an incident/injury

- 5 Very High Death / permanent incapacity / widespread loss
- 4 High Major Injury (Reportable Category) / Severe Incapacity / Serious Loss
- 3 Moderate Injury / illness of 3 days or more absence (reportable category) / Moderate loss
- 2 Slight Minor injury / illness – immediate First Aid only / slight loss
- 1 Negligible No injury or trivial injury / illness / loss

Likelihood value = what is the potential of an incident or injury occurring

- 5 Almost certain to occur
- 4 Likely to occur
- 3 Quite possible to occur
- 2 Possible in current situation
- 1 Not likely to occur

risk rating = severity value × likelihood value

risk ratings are classified as low (1 – 5), medium (6 – 9) and high (10 – 25)

Risk Classification and Actions:

Rating	Classification	Action
1 – 5	Low	Tolerable risk - Monitor and Manage
6 – 9	Medium	Review and introduce additional controls to mitigate to "As Low As Reasonably Practicable" (ALARP)
10 – 25	High	Stop work immediately and introduce further control measures

SEVERITY

1	2	3	4	5
1 Low	Low	Low	Low	Low
2 Low	Low	Medium	Medium	High
3 Low	Medium	Medium	High	High
4 Low	Medium	High	High	High
5 Low	High	High	High	High

LIKELIHOOD

8.3. Appendix 3– Project Plan

[illegible][illegible]

8.4. Appendix 4 -Kalman Filter Code 1 – Constant

`%Example from http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies`

```
zk = zeros (1,200);
```

```
y = 4 * ones (1,200);
```

```
for n=1:200
```

```
    zk (n) = 4 + 0.5*randn;
```

```
end
```

```
x0=0;
```

```
P0=1;
```

```
R=0.25;
```

```
A=1;
```

```
Q=0;
```

```
x = zeros (1,200);
```

```
k = zeros (1,200);
```

```
p = zeros (1,200);
```

```
k(1)= P0/(P0+R);
```

```
x(1)= x0 + k(1)*(zk(1)-x0);
```

```
p(1)= (1-k(1))*P0;
```

```
for t=2:200
```

```
    k(t)= p(t-1)/(p(t-1)+R);
```

```
    x(t)= x(t-1) + k(t)*(zk(t)-x(t-1));
```

```
    p(t)=(1-k(t))*p(t-1);
```

```
end
```

```
subplot(121)
```

```
plot(x)
```

```
hold on
```

```
plot (y, 'Color','r')
```

```
subplot(122)
```

```
plot(x-4)
```

8.5. Appendix 5 - Kalman Filter Code 2 – Linear

```
%An object travelling in 1D at a constant velocity of 1.5m/s

yk = zeros (1,200);
for n=1:200
    yk (n) = 1.5*n + 3*randn ;
end % creates 'measured' inputs with 'measurements' being independent
    %of each other i.e. erros don't propagate

R=1; %the function 'randn' ouputs normally distributed random numbers
    %this makes the standard deviation=1, therefore variance=1

X0=0; %starting at origin
P0=1; %any non-zero value otherwise K=0
A=1;
Q=0;
U=1.5; %travelling speed
W=0; %Assuming no white noise
H=1; %1 as just numbers not matrices

B = zeros (1,200);
for n=1:200
    B(n)= n;
end %for elapsed time

xkp = zeros (1,200);
x = zeros (1,200);
k = zeros (1,200);
pkp = zeros (1,200);
pk = zeros (1,200);

%t1 Predicted state
xkp(1)= A*X0 + B(1)*U + W;
pkp(1)= A*P0*A + Q;

%update w/ new measurements and kalman gain
k(1)=(pkp(1)*H)*inv(H*pkp(1)*H + R);
x(1)= xkp(1) + k(1)*(yk(1)-H*xkp(1));
pk(1)= (1-k(1)*H)*pkp(1);


for t=2:200
    %t(n) Predicted state
    xkp(t)= A*x(t-1) + 1*U + W;
    pkp(t)= A*pk(t-1)*A + Q;
    %update w/ new measurements and kalman gain
    k(t)=(pkp(t)*H)*inv(H*pkp(t)*H + R);
    x(t)= xkp(t) + k(t)*(yk(t)-H*xkp(t));

    pk(t)= (1-k(t-1)*H)*pkp(t-1);
end

test = linspace(0,300,200);

subplot(121)
plot(x)
hold on
plot (yk, 'Color','r')
subplot(122)
plot(x-test)
hold on
plot (yk-test, 'Color','r')
```


8.6. Appendix 6 - MPU 9250 Register Map

	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013
-----------------------------------------------------------------------------------	-----------------------------------------------	-----------------------------------------------------------------------------

3 Register Map for Gyroscope and Accelerometer

The following table lists the register map for the gyroscope and accelerometer in the MPU-9250 MotionTracking device.

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
00	0	SELF_TEST_X_GYRO	RAW	XG_ST_DATA [7:0]							
01	1	SELF_TEST_Y_GYRO	RAW	YG_ST_DATA [7:0]							
02	2	SELF_TEST_Z_GYRO	RAW	ZG_ST_DATA [7:0]							
0D	13	SELF_TEST_X_ACCEL	RAW	XA_ST_DATA [7:0]							
0E	14	SELF_TEST_Y_ACCEL	RAW	YA_ST_DATA [7:0]							
0F	15	SELF_TEST_Z_ACCEL	RAW	ZA_ST_DATA [7:0]							
13	19	XG_OFFSET_H	RAW	X_OFFSETS_USR [15:8]							
14	20	XG_OFFSET_L	RAW	X_OFFSETS_USR [7:0]							
15	21	YG_OFFSET_H	RAW	Y_OFFSETS_USR [15:8]							
16	22	YG_OFFSET_L	RAW	Y_OFFSETS_USR [7:0]							
17	23	ZG_OFFSET_H	RAW	Z_OFFSETS_USR [15:8]							
18	24	ZG_OFFSET_L	RAW	Z_OFFSETS_USR [7:0]							
19	25	SMP_LRT_DIV	RAW	SMP_LRT_DIV[7:0]							
1A	26	CONFIG	RAW	-	FIFO_MODE	EXT_SYNC_SET[2:0]			DLPF_CFG[2:0]		
1B	27	GYRO_CONFIG	RAW	XGYRO_Ct_en	YGYRO_Ct_en	ZGYRO_Ct_en	GYRO_FS_SEL [1:0]		-	FCHOICE_B[1:0]	
1C	28	ACCEL_CONFIG	RAW	ax_st_en	ay_st_en	az_st_en	ACCEL_FS_SEL[1:0]		-		
1D	29	ACCEL_CONFIG 2	RAW	-				ACCEL_FCHOICE_B		A_DLPF_CFG	
1E	30	LP_ACCEL_ODR	RAW	-				Lposc_cksel [3:0]			
1F	31	WOM_THR	RAW	WOM_Threshold [7:0]							
23	35	FIFO_EN	RAW	TEMP_FIFO_EN	GYRO_XO_UT	GYRO_YO_UT	GYRO_ZO_UT	ACCEL	SLV2	SLV1	SLV0
24	36	I2C_MST_CTRL	RAW	MULT_MST_EN	WAIT_FOR_ES	SLV_3_FIFO_EN	I2C_MST_P_NSR	I2C_MST_CLK[3:0]			
25	37	I2C_SLV0_ADDR	RAW	I2C_SLV0_RNW	I2C_ID_0 [5:0]						
26	38	I2C_SLV0_REG	RAW	I2C_SLV0_REG[7:0]							
27	39	I2C_SLV0_CTRL	RAW	I2C_SLV0_EN	I2C_SLV0_BYTE_SW	I2C_SLV0_REG_DIS	I2C_SLV0_GRP	I2C_SLV0_LEN[3:0]			
28	40	I2C_SLV1_ADDR	RAW	I2C_SLV1_RNW	I2C_ID_1 [5:0]						
29	41	I2C_SLV1_REG	RAW	I2C_SLV1_REG[7:0]							
2A	42	I2C_SLV1_CTRL	RAW	I2C_SLV1_EN	I2C_SLV1_BYTE_SW	I2C_SLV1_REG_DIS	I2C_SLV1_GRP	I2C_SLV1_LEN[3:0]			
2B	43	I2C_SLV2_ADDR	RAW	I2C_SLV2_RNW	I2C_ID_2 [5:0]						
2C	44	I2C_SLV2_REG	RAW	I2C_SLV2_REG[7:0]							
2D	45	I2C_SLV2_CTRL	RAW	I2C_SLV2_EN	I2C_SLV2_BYTE_SW	I2C_SLV2_REG_DIS	I2C_SLV2_GRP	I2C_SLV2_LEN[3:0]			
2E	46	I2C_SLV3_ADDR	RAW	I2C_SLV3_RNW	I2C_ID_3 [5:0]						
2F	47	I2C_SLV3_REG	RAW	I2C_SLV3_REG[7:0]							
30	48	I2C_SLV3_CTRL	RAW	I2C_SLV3_EN	I2C_SLV3_BYTE_SW	I2C_SLV3_REG_DIS	I2C_SLV3_GRP	I2C_SLV3_LEN[3:0]			
31	49	I2C_SLV4_ADDR	RAW	I2C_SLV4_RNW	I2C_ID_4 [5:0]						
32	50	I2C_SLV4_REG	RAW	I2C_SLV4_REG[7:0]							
33	51	I2C_SLV4_DO	RAW	I2C_SLV4_DO[7:0]							
34	52	I2C_SLV4_CTRL	RAW	I2C_SLV4_EN	SLV4_DONE_INT_EN	I2C_SLV4_REG_DIS	I2C_MST_DLY[4:0]				

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
35	53	I2C_SLV4_DI	R	I2C_SLV4_D[7:0]							
36	54	I2C_MST_STATUS	R	PASS THROUGH	I2C_SLV4 _DONE	I2C_LOST _ARB	I2C_SLV4 _NACK	I2C_SLV3 _NACK	I2C_SLV2 _NACK	I2C_SLV1 _NACK	I2C_SLV0 _NACK
37	55	INT_PIN_CFG	RAW	ACTL	OPEN	LATCH _INT_EN	INT_ANYR D _2CLEAR	ACTL_FSY NC	FSYNC _INT_MOD E_EN	BYPASS _EN	-
38	56	INT_ENABLE	RAW	-	WOM_EN	-	FIFO _OFLOW _EN	FSYNC_INT _EN	-	-	RAW_RDY _EN
3A	58	INT_STATUS	R	-	WOM_INT	-	FIFO _OFLOW _INT	FSYNC_INT	-	-	RAW_DATA _RDY_INT
3B	59	ACCEL_XOUT_H	R	ACCEL_XOUT_H[15:8]							
3C	60	ACCEL_XOUT_L	R	ACCEL_XOUT_L[7:0]							
3D	61	ACCEL_YOUT_H	R	ACCEL_YOUT_H[15:8]							
3E	62	ACCEL_YOUT_L	R	ACCEL_YOUT_L[7:0]							
3F	63	ACCEL_ZOUT_H	R	ACCEL_ZOUT_H[15:8]							
40	64	ACCEL_ZOUT_L	R	ACCEL_ZOUT_L[7:0]							
41	65	TEMP_OUT_H	R	TEMP_OUT_H[15:8]							
42	66	TEMP_OUT_L	R	TEMP_OUT_L[7:0]							
43	67	GYRO_XOUT_H	R	GYRO_XOUT_H[15:8]							
44	68	GYRO_XOUT_L	R	GYRO_XOUT_L[7:0]							
45	69	GYRO_YOUT_H	R	GYRO_YOUT_H[15:8]							
46	70	GYRO_YOUT_L	R	GYRO_YOUT_L[7:0]							
47	71	GYRO_ZOUT_H	R	GYRO_ZOUT_H[15:8]							
48	72	GYRO_ZOUT_L	R	GYRO_ZOUT_L[7:0]							
49	73	EXT_SENS_DATA_00	R	EXT_SENS_DATA_00[7:0]							
4A	74	EXT_SENS_DATA_01	R	EXT_SENS_DATA_01[7:0]							
4B	75	EXT_SENS_DATA_02	R	EXT_SENS_DATA_02[7:0]							
4C	76	EXT_SENS_DATA_03	R	EXT_SENS_DATA_03[7:0]							
4D	77	EXT_SENS_DATA_04	R	EXT_SENS_DATA_04[7:0]							
4E	78	EXT_SENS_DATA_05	R	EXT_SENS_DATA_05[7:0]							
4F	79	EXT_SENS_DATA_06	R	EXT_SENS_DATA_06[7:0]							
50	80	EXT_SENS_DATA_07	R	EXT_SENS_DATA_07[7:0]							
51	81	EXT_SENS_DATA_08	R	EXT_SENS_DATA_08[7:0]							
52	82	EXT_SENS_DATA_09	R	EXT_SENS_DATA_09[7:0]							
53	83	EXT_SENS_DATA_10	R	EXT_SENS_DATA_10[7:0]							
54	84	EXT_SENS_DATA_11	R	EXT_SENS_DATA_11[7:0]							
55	85	EXT_SENS_DATA_12	R	EXT_SENS_DATA_12[7:0]							
56	86	EXT_SENS_DATA_13	R	EXT_SENS_DATA_13[7:0]							
57	87	EXT_SENS_DATA_14	R	EXT_SENS_DATA_14[7:0]							
58	88	EXT_SENS_DATA_15	R	EXT_SENS_DATA_15[7:0]							
59	89	EXT_SENS_DATA_16	R	EXT_SENS_DATA_16[7:0]							
5A	90	EXT_SENS_DATA_17	R	EXT_SENS_DATA_17[7:0]							
5B	91	EXT_SENS_DATA_18	R	EXT_SENS_DATA_18[7:0]							
5C	92	EXT_SENS_DATA_19	R	EXT_SENS_DATA_19[7:0]							
5D	93	EXT_SENS_DATA_20	R	EXT_SENS_DATA_20[7:0]							
5E	94	EXT_SENS_DATA_21	R	EXT_SENS_DATA_21[7:0]							
5F	95	EXT_SENS_DATA_22	R	EXT_SENS_DATA_22[7:0]							
60	96	EXT_SENS_DATA_23	R	EXT_SENS_DATA_23[7:0]							
63	99	I2C_SLV0_DO	RAW	I2C_SLV0_DO[7:0]							
64	100	I2C_SLV1_DO	RAW	I2C_SLV1_DO[7:0]							
65	101	I2C_SLV2_DO	RAW	I2C_SLV2_DO[7:0]							

InvenSense	MPU-9250 Register Map and Descriptions	Document Number: RM-MPU-9250A-00 Revision: 1.4 Release Date: 9/9/2013
-------------------	-----------------------------------------------	-----------------------------------------------------------------------------

Addr (Hex)	Addr (Dec.)	Register Name	Serial I/F	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
66	102	I2C_SLV3_DO	RAW	I2C_SLV3_DO[7:0]							
67	103	I2C_MST_DELAY_CTRL	RAW	DELAY_ES_SHADOW	-	-	I2C_SLV4_DLY_EN	I2C_SLV3_DLY_EN	I2C_SLV2_DLY_EN	I2C_SLV1_DLY_EN	I2C_SLV0_DLY_EN
68	104	SIGNAL_PATH_RESET	RAW	-	-	-	-	-	GYRO_RST	ACCEL_RST	TEMP_RST
69	105	MOT_DETECT_CTRL	RAW	ACCEL_INT_EL_EN	ACCEL_INT_EL_MODE	-			-		
6A	106	USER_CTRL	RAW	-	FIFO_EN	I2C_MST_EN	I2C_IF_DIS	-	FIFO_RST	I2C_MST_RST	SIG_COND_RST
6B	107	PWR_MGMT_1	RAW	H_RESET	SLEEP	CYCLE	GYRO_STANDBY	PD_PTAT	CLKSEL[2:0]		
6C	108	PWR_MGMT_2	RAW	-		DIS_XA	DIS_YA	DIS_ZA	DIS_XG	DIS_YG	DIS_ZG
72	114	FIFO_COUNTH	RAW	-			FIFO_CNT[12:8]				
73	115	FIFO_COUNTL	RAW	FIFO_CNT[7:0]							
74	116	FIFO_R_W	RAW	D[7:0]							
75	117	WHO_AM_I	R	WHOAMI[7:0]							
77	119	XA_OFFSET_H	RAW	XA_OFFSETS [14:7]							
78	120	XA_OFFSETSET_L	RAW	XA_OFFSETS [6:0]							-
7A	122	YA_OFFSETSET_H	RAW	YA_OFFSETS [14:7]							
7B	123	YA_OFFSETSET_L	RAW	YA_OFFSETS [6:0]							-
7D	125	ZA_OFFSETSET_H	RAW	ZA_OFFSETS [14:7]							
7E	126	ZA_OFFSETSET_L	RAW	ZA_OFFSETS [6:0]							-

Table 1 MPU-9250 mode register map for Gyroscope and Accelerometer

Note: Register Names ending in _H and _L contain the high and low bytes, respectively, of an internal register value.

In the detailed register tables that follow, register names are in capital letters, while register values are in capital letters and italicized. For example, the ACCEL_XOUT_H register (Register 59) contains the 8 most significant bits, *ACCEL_XOUT[15:8]*, of the 16-bit X-Axis accelerometer measurement, *ACCEL_XOUT*.

The reset value is 0x00 for all registers other than the registers below.

- Register 107 (0x01) Power Management 1
- Register 117 (0x71) WHO_AM_I

8.7. Appendix 7 - IMU Code to obtain raw values

```
// code modified from https://www.youtube.com/watch?v=M9IZ5Qy5S2s
#include <Wire.h>
long accelX, accelY, accelZ; //accelerometer
long gyroX, gyroY, gyroZ;//gyro

void setup() {
  Serial.begin(9600);
  Wire.begin(); // starting I2C communication

  // initialising the sensor //SETTING UP POWER
  Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
  Wire.write(0x6B); // Power Management 1
  Wire.write(0x00); // pg 40
  Wire.endTransmission();

  Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
  Wire.write(0x6C); // Power Management 2
  Wire.write(0x00); // pg 41 - enables gyro and acc x,y,z
  Wire.endTransmission();

  //GYRO CONFIGURATION
  Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
  Wire.write(0x1B); // gyro configuration
  Wire.write(0x00); // pg 14 - sets the full scale to +/- 250 degress/second
  Wire.endTransmission();

  //ACC CONFIGURATION
  Wire.beginTransmission(0x68); //I2C address of the MPU (as SJ2 is in place)
  Wire.write(0x1C); // acc configuration
  Wire.write(0x00); // pg 14 - sets the full scale to +/- 2gs
  Wire.endTransmission();
}

void loop() {
  //get raw data (does not represent gs or dps, needs to be scaled depending on setup)
```

```

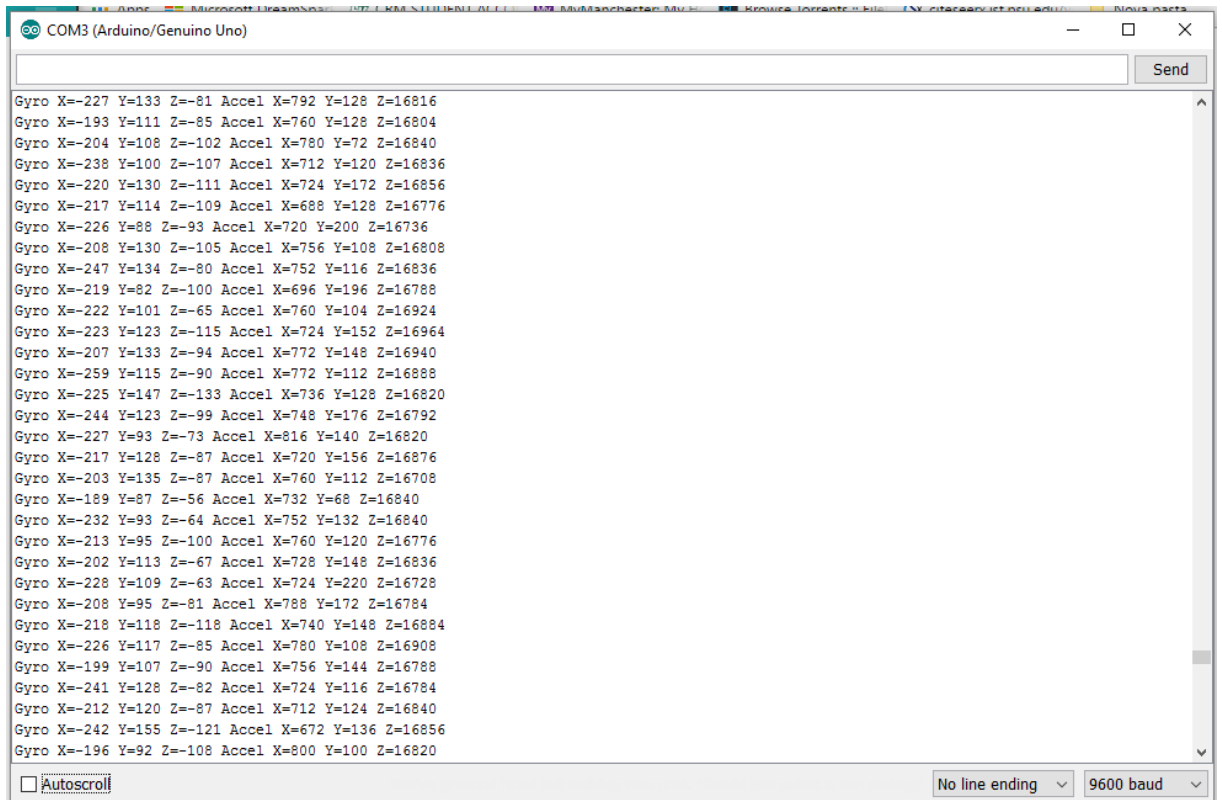
//accelerometer readings
Wire.beginTransmission(0x68); //I2C address of the MPU
Wire.write(0x3B); //Starting register for Accel Readings
Wire.endTransmission();
Wire.requestFrom(0b1101000,6); //Request Accel Registers (3B - 40)
while(Wire.available() < 6);
accelX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
accelY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
accelZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ


//gyro data
Wire.beginTransmission(0x68); //I2C address of the MPU
Wire.write(0x43); //Starting register for Gyro Readings
Wire.endTransmission();
Wire.requestFrom(0b1101000,6); //Request Gyro Registers (43 - 48)
while(Wire.available() < 6);
gyroX = Wire.read()<<8|Wire.read(); //Store first two bytes into accelX
gyroY = Wire.read()<<8|Wire.read(); //Store middle two bytes into accelY
gyroZ = Wire.read()<<8|Wire.read(); //Store last two bytes into accelZ


Serial.print("Gyro");
Serial.print(" X=");
Serial.print(gyroX);
Serial.print(" Y=");
Serial.print(gyroY);
Serial.print(" Z=");
Serial.print(gyroZ);
Serial.print(" Accel");
Serial.print(" X=");
Serial.print(accelX);
Serial.print(" Y=");
Serial.print(accelY);
Serial.print(" Z=");
Serial.println(accelZ);
}

```

8.8. Appendix 8 – IMU Output



The screenshot shows the Arduino IDE Serial Monitor window for a COM3 (Arduino/Genuino Uno) connection. The window displays a continuous stream of IMU data in the following format: Gyro X=... Y=... Z=... Accel X=... Y=... Z=... The data is scrolling upwards, and the 'Autoscroll' checkbox is checked. At the bottom right, the settings are set to 'No line ending' and '9600 baud'.

```
Gyro X=-227 Y=133 Z=-81 Accel X=792 Y=128 Z=16816
Gyro X=-193 Y=111 Z=-85 Accel X=760 Y=128 Z=16804
Gyro X=-204 Y=108 Z=-102 Accel X=780 Y=72 Z=16840
Gyro X=-238 Y=100 Z=-107 Accel X=712 Y=120 Z=16836
Gyro X=-220 Y=130 Z=-111 Accel X=724 Y=172 Z=16856
Gyro X=-217 Y=114 Z=-109 Accel X=688 Y=128 Z=16776
Gyro X=-226 Y=88 Z=-93 Accel X=720 Y=200 Z=16736
Gyro X=-208 Y=130 Z=-105 Accel X=756 Y=108 Z=16808
Gyro X=-247 Y=134 Z=-80 Accel X=752 Y=116 Z=16836
Gyro X=-219 Y=82 Z=-100 Accel X=696 Y=196 Z=16788
Gyro X=-222 Y=101 Z=-65 Accel X=760 Y=104 Z=16924
Gyro X=-223 Y=123 Z=-115 Accel X=724 Y=152 Z=16964
Gyro X=-207 Y=133 Z=-94 Accel X=772 Y=148 Z=16940
Gyro X=-259 Y=115 Z=-90 Accel X=772 Y=112 Z=16888
Gyro X=-225 Y=147 Z=-133 Accel X=736 Y=128 Z=16820
Gyro X=-244 Y=123 Z=-99 Accel X=748 Y=176 Z=16792
Gyro X=-227 Y=93 Z=-73 Accel X=816 Y=140 Z=16820
Gyro X=-217 Y=128 Z=-87 Accel X=720 Y=156 Z=16876
Gyro X=-203 Y=135 Z=-87 Accel X=760 Y=112 Z=16708
Gyro X=-189 Y=87 Z=-56 Accel X=732 Y=68 Z=16840
Gyro X=-232 Y=93 Z=-64 Accel X=752 Y=132 Z=16840
Gyro X=-213 Y=95 Z=-100 Accel X=760 Y=120 Z=16776
Gyro X=-202 Y=113 Z=-67 Accel X=728 Y=148 Z=16836
Gyro X=-228 Y=109 Z=-63 Accel X=724 Y=220 Z=16728
Gyro X=-208 Y=95 Z=-81 Accel X=788 Y=172 Z=16784
Gyro X=-218 Y=118 Z=-118 Accel X=740 Y=148 Z=16884
Gyro X=-226 Y=117 Z=-85 Accel X=780 Y=108 Z=16908
Gyro X=-199 Y=107 Z=-90 Accel X=756 Y=144 Z=16788
Gyro X=-241 Y=128 Z=-82 Accel X=724 Y=116 Z=16784
Gyro X=-212 Y=120 Z=-87 Accel X=712 Y=124 Z=16840
Gyro X=-242 Y=155 Z=-121 Accel X=672 Y=136 Z=16856
Gyro X=-196 Y=92 Z=-108 Accel X=800 Y=100 Z=16820
```