THE DESIGN, MODELLING, CONSTRUCTION AND TESTING OF A TWO WHEELED SELF-BALANCING ROBOT

Thesis · April 2019

CITATIONS
0

READS
2,063

1 author:

Joshua Goring
University of Strathclyde
1 PUBLICATION **0** CITATIONS

SEE PROFILE

School of Engineering and the Built Environment

Honours Project (Engineering)

MHH623549

Project Title: THE DESIGN, MODELLING, CONSTRUCTION AND TESTING OF A TWO WHEELED SELF-BALANCING ROBOT

Programme/ Option: Mechanical Electronic Systems Engineering

Author: Joshua Thomas Goring

Student Number: S1509656

Supervisor(s): George Cullen

Non-Disclosure Agreement. ~~Yes~~ / **No.**

Except where explicitly stated all work in this report, including the appendices, is my own

Signed: JOSHUA THOMAS GORING Date: 25.04.19

Abstract

The control of a self-balancing robot is a popular topic and the theory behind it is becoming increasingly used and developed in transportation systems and robotics. This paper looks at the design, control, and manufacture of a self-balancing robot. This robot is equipped with an accelerometer, gyroscope, and motor encoders that are used with the goal of balancing the robot.

In addition to this, the paper looks the area mathematically modelling the robot and the route of simulation.

The system was designed to use angular data to control the motor speed accordingly. This data is used to provide pulse width modulation data to the motors in order to bring the robot's angle back to 0.

Acknowledgments

I would like to thank my lecturer and project supervisor George Cullen for helping me throughout this project and inspiring me to further my control theory.

As well as this I would like to thank my mother Ruth Moss for her advice and support.

Symbols and Abbreviations

Symbols

F	Force (N)
N	Nominal Force (N)
x	Position of wheels (m)
M	Mass (Kg)
φ	Angle between upright position and pendulum (°)
θ	Angle between y axis and pendulum (°)
g	Gravitational constant (m/s)
l	Length to centre of mass (m)
I	Inertia (m/s^2)
r	Radius of the Wheel (m)

Abbreviations

CMG	Control moment gyroscope
CAD	Computer Aided Design
Matlab	Matrix Laboratory
PWM	Pulse width modulation
SISO	Single input single output

Table of Contents

1. Introduction

1.1 Background

A two wheeled self-balancing robot can be defined as a modern-day inverted pendulum. Inverted pendulums are seen everywhere and a self-balancing robot can easily be compared to a human body where the human body is seen as a pendulum pivoting around the ankle joint. Self-balancing systems in the modern era are increasingly becoming more popular, often being used as transportation methods and with companies such as Segway emerging in the market, two wheeled balancing systems are taking the world by storm.

Furthermore, these two wheeled balancing systems are paving the way for new robotics, specifically robots that specialise in assistance, due to their ease of mobility and functionality.

A good example of this is the HITACHI EMIEW (see Figure 1), a robot designed on a two-wheel self-balancing system. The robot can move in compact spaces and can detect and avoid obstacles. Designed for human interaction, the robot can talk to humans and even interact with them using robotic arm features.

Figure 1 - HITACHI EMIEW 1

1.2 Project Aims

The purpose of this project is to design and create a self-balancing robot. Although this is not a new concept, it is one that has excited hobbyists to advanced control lecturers.

In addition, as self-balancing robots are becoming more popular in the use of robotics, as shown by the HITACHI EMIEW, it could be argued that there is a need to research and developed this topic.

Therefore, I am interested in exploring the concept of two wheeled self-balancing robots, as a way to better understand the engineering and the control systems involved in the project.

This process starts with the component selection, construction of a physical frame for the robot, then moving on to the mathematical modelling and the implementation of a control system - using Simulink - that could be used to effectively balance a two wheeled robot.

In addition to this the system would be modelled to give transfer functions that could be used in Simulink and MATLAB's SISO (single input single output) tool to give readings as to how system operates and how it can be improved.

1.2 Theory

The theory of the self-balancing robot was heavily based around the theory of an inverted pendulum. As shown below (see Figure 2 Inverted pendulum on cart (Modern Control Engineering by Ogata)), the inverted pendulum model is made up of a weighted rod on a moving cart. The basic control principle states that, whilst the weight of the pendulum is directly above the cart parallel to the y axis the pendulum will balance. However, as the system is unstable the rod will begin to fall. The cart will move in the direction the pendulum is falling, to correct the position of the pendulum and return it to centre. Essentially this keeps the pendulum upright.

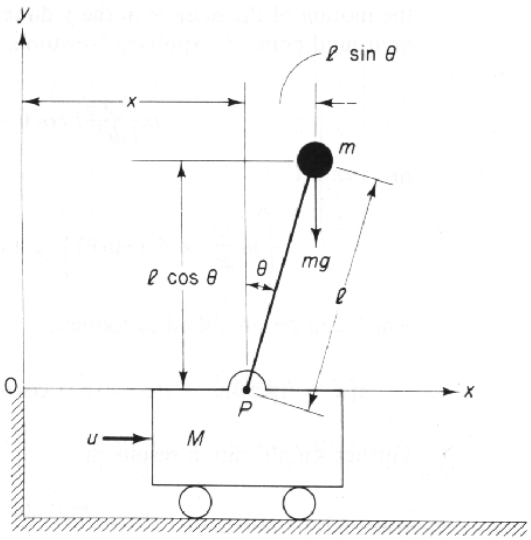


Figure 2 Inverted pendulum on cart (Modern Control Engineering by Ogata)

The same can be constructed of the self-balancing robot where wheels, connected to two DC motors, are treated as the cart, in the inverted pendulum model.

As the robot is unstable, when left, the robot is going to fall forwards or backward. The motors can be used to drive in the direction of the falling pendulum in order to catch its fall and balance the system.

This can be described by the following commonly used flow chart:

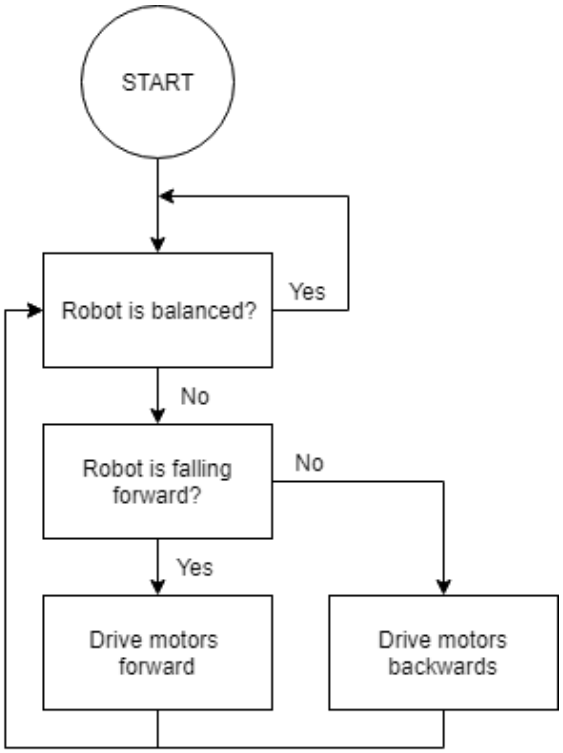


Figure 3

The project was heavily based around the MATLAB Inverted Pendulum: System Modelling tutorial (Control Tutorials for Matlab and Simulink. 2017) where the transfer functions and state space models were calculated for an inverted pendulum attached to a motorised cart. The same method used to obtain these transfer functions were used in the following project and used to help mathematically model and validate the system.

1.3 Objectives

To effectively complete the project, the following methodology was implemented:

- Literature review
- Investigation of components
- Design of physical structure
- Calibration of components
- Mathematical modelling and creation of transfer functions
- Closed loop Simulink model
- Tuning of model
- Creation of a program to run on the physical model
- Testing of the model and concluding results

2. Literature Review

To inform the project, a literature search was undertaken using the following databases:

Discover

Pro Quest

Google scholar

The initial keywords used were; *self-balancing, two wheeled, control systems, robot** and *inverse pendulum*.

94 journal articles were returned from these searches. Each article was reviewed and a secondary search was undertaken using the keywords; *gyroscope, accelerometer, MATLAB, Simulink,* and *sensors.* 85 articles were returned from this search. Peer reviewed journal articles were exclusively selected, due to their increased likelihood of robust and valid research results. Papers that were not in double-blind, peer reviewed journals were excluded.

The papers were limited by a date range of 2013 – present day, as more modern technologies were deemed more valuable. An exception was made for one paper from 1996 as it proved to be a helpful source for mathematically modelling the robot and was seen to be a seminal work.

Duplicates were excluded, as were articles focussing on humanoid robots, as these were not relevant to this project. Articles that didn't use gyroscopes and accelerometers, for example used optical sensors, were also excluded, on the basis of lack of relevance to this topic.

Although the self-balancing robot has proven to be a popular topic, and therefore a lot of research was obtained, after examining the articles in depth, 10 articles were reviewed. Each of these demonstrated different methods on how to achieve balance from a two wheeled robot.

The literature reviewed examined the hardware used in the design of the robot, the software and control systems used and some of the mathematical terminology. As well as this, some alternate ways to balance the robot such as the use of a CMG were reviewed.

a) Many of the papers looked at using a gyroscope, accelerometer, motor encoders or a combination of these sensors to work out the angle. In the paper titled "Trajectory tracking control for navigation of the inverse pendulum type self-contained robot" (Ha & Yuta, 1996), the research looked at designing and modelling a robot that could independently move and navigate itself whilst maintaining stability and balance.

The robot was made of two wheels independent of each other, rotary encoders and a gyroscope to measure the angular velocity of the body.
The system was first modelled to maintain balance in one plane. A control system was introduced to enable the robot change direction but when this was implemented into the programming of the robot, it affected its balancing control system. Therefore, the steering control and the balance control had to be separated. After this the robot's movement was tested in stages. Firstly, the robot's balance was tested. Subsequently, the robots turning was tested. Then the robot was tested for trajectory tracking and positional control. Lastly, the robot was tested for indoor navigation, to show its overall performance and validity of the command system.

b) Another paper that made use of a gyroscope and motor encoders was a paper titled "Two Wheel Self Balancing Robot" (Riasat, Iqbal, Adeel & Khan, 2013).

In this paper, the basic design and breakdown of a two wheeled, self-balancing robot was discussed. The research broke down the process from building the structure, mathematically modelling the system and creating the PID controller in MatLab. The research simplified the process of correcting the robot, by stating that it could only fall forwards or backwards and therefore, to correct it the motorised wheels would spin either clockwise or anti-clockwise to prevent the structure from toppling over.
The robot was made using a stacked shelf design where the electronics can be placed. It had two motors (connected to a H Bridge) to drive the wheels and a microcontroller to process the instructions given by the programming. The robot was also equipped with two sensors – an accelerometer and optical encoders in the motors – that gave a value for tilt. These two sensors calculated the error of the signal and fed it to the PID controller which generated a control signal to correct the system. The dynamics of the two-wheeled robot were the worked out and the system was modelled. Matlab was used to program the model, due to its ease of analysing and designing the system. A PID controller was created and simulated using Matlab. The PID controller judged the error of the system and generated pulse width modulations, which were used to maintain the robots balance.
The system successfully managed to balance itself but was not able to be controlled to move backwards and forwards. It was worth noting the stacked design of the robot in this paper as it proved effective for placing electronics.

The following papers largely focused on the use of a gyroscope and motor encoders.

c) The paper “Performance Evaluation of MMA7260QT and ADXL345 on Self Balancing Robot” (Ferdinando, Khoswanto & Purwanto, 2013) looked in more depth at the use of a gyroscope and accelerometer and examined how to work out tilt angle.

The paper started by discussing how two wheeled self-balancing robots have become popular transport options for urban commuters. It stated that ‘all self-balancing robots must detect the angle of inclination of the platform the gyroscope is mounted to’.
The paper used a gyroscope and an accelerometer to detect the tilt position of the robot. The accelerometer measured acceleration and when detecting the effects of gravity, the accelerometer detected 1g when parallel and 0g when perpendicular to the accelerometer’s axis. If the angle was in-between these two points, then trigonometry could be used to work out acceleration. The problem with the accelerometer in this case, was that the signal was masked by a lot of noise. To accommodate for this, a gyroscope was also inserted into the system, to help calculate the angle of inclination. Whilst this was meant to solve the problem, it was found that the gyroscope’s readings for angle experienced drift. To accommodate for these problems, a Kalman filter was used to combine the signals. This produced a more accurate reading for the angle of inclination.
The robot was then rotated using a servo motor and the angle on inclination was measured. For this experiment, an Arduino uno board was used as a microcontroller. The Kalman filter managed to estimate the angle correctly and reduce the noise.

d) The paper “Modeling of a Two-Wheeled Self-Balancing Robot Driven by DC Gearmotors” (Frankovský et al., 2017) looked at electrical parameters and made a comparison to the mechanical parameters of the system to work out tilt angle.

The paper examined the modelling of a self-balancing robot consisting of a robot’s body and two wheels. It stated that the robot could be modelled in the same way as an inverted pendulum. The researchers modelled the robot, ignoring the conditions of slip and used Lagrange’s second order equations in the modelling process. The robot used brushed, DC motors to act as the actuators that drove the wheels, required to help the robot move and balance. It was found that there was a proportional relationship between the electrical parameters such as the current and back EMF, in the actuator subsystem, to motor torque and angular velocity. Using the values of back EMF to work out angular velocity and current to work out motor torque is useful, as these values can be used to correct

the robots balance. From these relationships, an equation for the overall DC motor characteristics was formed. To provide more torque to the wheels, a gearbox was used. To balance the robot, it was determined that movement of the wheels was required, and the research made use of a state feedback controller, where the values of back EMF and current were used to measure errors, in order to feedback into the system and stabilise the robot.

It was found in certain papers that a common problem, that occurred when designing a two wheeled self-balancing robot, was external disturbances. These disturbances would be the change in the floor's inclination, to wind or even bumps in terrain. To tackle this problem, some articles looked at advanced ways to predict disturbances before the robot encountered them.

e) The paper titled "Disturbance rejection using feed-forward control system on self-balancing robot" (Henryranu Prasetio & Kurniawan, 2018) looked at the use of a "feed forward control system" in a Segway style device.

This system took advantage of two ultrasonic sensors that aimed to detect uneven ground and disturbance in terrain. The Segway then took this information and altered the velocity and angle of the Segway accordingly. For this to work, the control system had to be fine-tuned, so that the velocity and angle of the Segway adjusted at the correct time to overcome the disturbances. The System also used an "Ensemble Kalman Filter". This was used to remove noise measurement from the accelerometer and deviation measurement from the gyroscope, (similar to in the paper "Performance Evaluation of MMA7260QT and ADXL345 on Self Balancing Robot" (Ferdinando, Khoswanto & Purwanto, 2013))

The disturbance was computed through a transfer function that considers; Dead function time, gain and lead lag. The project group worked out the Dead time function "Tdt" as the "delay time between the disturbances with the output controller to dampen the disturbance". They then applied a gain to the feed forward controller and labelled it "Kff". The lead time "Tld" was worked out "as the delay time between output controllers with a process variable output from the plant." Lastly the lag time "Tlg" was noted. This was "the delay time between the emerging disturbances with the process variable output of the plant". The ultrasonic sensors were then attached and experimental values for "Ttd", "Tld" and "Tlg" were found. The system was coupled with the "Ensemble Kalman filter" EnKF at this point.

After these values were obtained, a reasonable value of gain was obtained through trial and error where the disturbance signal had to be mirrored by the output value of the controller.

Subsequently, the Segway was tested on two scopes, one incline and one decline and the value for gain (Kff) was further refined and the optimum value during these experiments were found.

f) Another paper that looked at solving the problem of disturbances before they happened was "Modelling, analysis and compensation of disturbances during task execution of a wheeled inverted pendulum type assistant robot using a unified controller" (Canete & Takahashi, 2015).

This paper stated that using robots to help humans with tasks, usually requires a high level of torque, which means that they are not suitable for use around humans. This project aimed to create a two wheeled self-balancing robot that used its own weight as leverage, to create large lifting forces. Therefore, the robot was tested in various scenarios and given loads to carry. An extended state observer (ESO) was applied to the system, to accommodate for disturbances, as it was found that they could not be measured or known beforehand. The model was stabilised using an augmented linear quadratic regulator that used terms based on the extended state observer, to reduce disturbance. A disturbance test was created to test the compensation ability of the robot. This test was created by providing a ramp for the robot to climb up. Next, a load was also added to the robot, to test the effectiveness of the compensation terms provided by the ESO.
As the robot was designed to initialise from a sitting down position, it was tested to bring itself up to a standing position. This turned out to be successful and the robot was able to keep its balance. The key to this success was that the robot treated all system changes as disturbances and then compensated for their effects.

The modelling of the robot and its control system varied from paper to paper.

g) In the paper "Comparison of different control theories on a two wheeled self-balancing robot" (Rahman, Rashid, Hassan & Hossain, 2018) the researchers investigated using three of the most common control theories within self-balancing robot technology, as well as different programs used to run them.

The paper started off discussing 'Robot Operating System' (ROS) and 'Gazebo', two of the leading software's relating to robotics. These programs are open source and therefore contain many libraries and tools that can be used for robotic development.
The first test of the robot was created in link. The key measurements of the robot were found by integrating a CAD model of the robot into the program. Once the model was set up, it was tested using PID, fuzzy logic and LQR. It was found that the PID controller gave the most stable performance, the LQR gave the fastest response curve and that using the fuzzy logic controller made the robot highly

unstable. The paper also discussed the use of MATLAB and Simulink to program the robot but stated that ‘there is a high chance of error’ when mathematically modelling the device and that ‘assumptions have to be made about the laws of physics’. Therefore, it suggested using Gazebo as the physics engine derived the mathematical model from the robot’s properties.

h) Although MATLAB was not suggested in the previous paper, the paper “Simplifications: The Rule Base for Stabilization of Inverted Pendulum System” (Hanafy & Metwally, 2014) looked at how to design a controller that could stabilise an inverted pendulum on a cart and made use of MATLAB and Simulink.

The experiment involved using fuzzy logic to stabilise the pendulum, as well as using its potential to overcome disturbances and external impact affecting the pendulum. The system used a fuzzy logic system, as they are quoted to be ‘convenient for qualitive system modelling’ and ‘they are very simple conceptually’.
The control system used in this paper makes use of MATLAB and Simulink. The paper found that the angle and the position of the cart could be effectively controlled using ‘neuro fuzzy controller, supported by learning techniques derived from neural networks’.

i) Another paper that made use of MATLAB was ‘Modeling, Simulation, and Optimal Control for Two-Wheeled Self-Balancing Robot’ (Modestus Oliver Asali, Ferry Hadary & Bomo Wibowo Sanjaya, 2017).

In this paper, the theory of pendulum on a cart was discussed. Instead of trying to balance the robot through trial and error, experiments or simple mathematical models, the model was fully mathematically designed.

The model was then analysed. A linear quadratic regulator (LQR) was applied, as a controller for the system. An LQR is a control theory that utilises state-space equations to analyse a system. The system was set up and simulated on Simulink and Matlab. It was found that the LQR was effective at controlling both the pitch and heading angle of the robot without falling. Therefore, the paper suggested using a non-linear controller to balance the robot and went on further, to discuss how it may have made the system more robust.

j) The final problem that was examined when reviewing the literature, was to do with the robot correcting itself. The paper “Development of a self-balancing robot with a control moment gyroscope” (Park & Cho, 2018) made a point of stating that when a self-balancing robot encounters a disturbance the robot will lose its stability and must move in order to correct this.

This paper stated that it was typical that ‘the larger the disturbance, the more the robot moves’. Put simply, in order for a robot to keep its balance, it must move to accommodate for the disturbance. This can be a problem when operating a self-balancing device in small areas or in areas of danger as when the robot corrects itself it could put itself in more risk. For example, it could drive into a wall or over a drop.

As well as the robot’s normal, self-balancing programming (using two motors attached to the wheels via a belt drive), and the robot was also equipped with 2 Control Moment Gyroscopes. This was to try and improve on its stability and safety. These CMG modules were made up of actuator motors that spun flywheels, in order to generate torque. The torque generated was directed perpendicularly to the rotational axis of the flywheel and gimbal. To ensure a constant speed of each flywheel, rotary encoders were implemented into the design of the CMG.

The robot was set up with an inertia measure unit. This consisted of an inclinometer and a gyroscope. The inclinometer was used to measure the tilt in situations of slow rotation. However, during situations where there was fast rotation, a gyroscope was needed to judge the angle of the body. The gyroscope measured the angle of the body by first measuring, and then integrating, the angular velocity. In the modelling of the self-balancing robot, only the sagittal plane was considered. This was because the motions of the robot in this experiment were mainly back and forth. ‘When a disturbance occurs, the CMG module of the KUWAY generates a reaction torque that corresponds to the disturbance.’

The robot was tested standing up and proved to keep its balance without the use of the CMG. The robot’s wheels were then locked, and it was tested using just the CMG module. It also managed to keep its balance with a small variation in tilt angle.

The robot was set up in an experiment to see the difference that the CMG would make when disturbance was applied. The robot was tested with and without the use of the CMG and it was found that by using a CMG, the distance the robot moved to correct itself was reduced. It was also found that the more torque produced the better the results proved.

Conclusion

Within the literature, when it came to detecting the tilt angle of the robot, the most effective and used method was to use a combination of an accelerometer and a gyroscope. This proved even more effective when put through a KALMAN filter. This method of discovering tilt angle proved effective and so it was decided that it was to be used as the method of working out tilt angle in the project.

As motor encoders also proved to be good at determining tilt angle it was decided that motors with motor encoders were to be used in the project.

The use of MATLAB and SIMULINK proved to be popular in the reviewed papers and so it was decided to be the main program for the robots programming. MATLAB and SIMULINK were stated to give “high chances of error” due to the fact they cannot mathematically model the robot accurately and so it was decided to compare the mathematical model with one of that made using CAD software or a physics engine.

When designing the model of the robot, it was mentioned that a stacked design proved effective due to the fact that it can be made tall and electronics can be stacked on the shelves. This design was to be used in the project.

3. Methodology

3.1 Component Selection

To effectively balance the robot, components had to be used. To interface these components, a micro controller that could fit to the construction of the robot, had to be used. In this project, an Arduino Due was used, due to its compatibility with both relevant hardware and software.

To effectively balance the robot, certain values had to be obtainable. These values are as follows:

- Angular Data of the pendulum (body of the robot)
- Position of the wheels

For this project, it was decided that a gyroscope and a 3-axis accelerometer would be used to measure the angular data. Where the accelerometer was set to measure angle and the gyroscope to measure angular rate. Motor encoders would then be used to measure the position of the wheels.

An accelerometer measures the force of gravity. When the accelerometer lies flat the x axis will read 0g where the y axis will measure 1g.
As the accelerometer starts to tilt, the x axis will start to be affected by gravity and will start reading a positive or negative value, depending upon the direction of tilt. When the accelerometer is rotated slowly until the y axis lies flat, the value of the y reading will move from 1 to 0g and the value on the x axis will move from 0 to 1g.

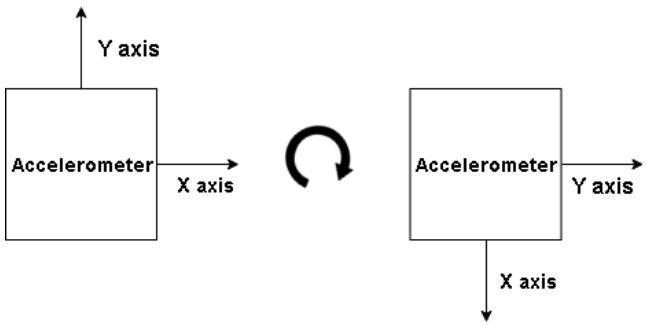


Figure 4 - Basic model of an accelerometer. Accelerometer with x = 0g y = 1g (left), accelerometer with x = 1g y = 0g (right)

It was found that it was not necessary to find angular data from the y and z axis. Only the x axis data was necessary. This is due to the fact that the y axis is not as sensitive as the x axis to changes in angle. It was found that the y axis data combined with the x axis data could be used to work out the angle through the use of trigonometry. However, this was deemed unnecessary and too difficult to implement.

A gyroscope was used to measure the speed of rotation of the robot, otherwise known as the angular rate.

When the robot is falling and the gyroscope is rotating, the gyroscope will read a value appropriate to speed and direction. If the robot is completely stationary the gyroscope will read 0. A gyroscope was not needed to measure in another plane as the robot only falls in one place.

3.1.1 Arduino Due

The Arduino Due (Figure 5 - Arduino Due) is a popular micro controller that is used in a lot of larger scale Arduino projects. The Arduino line of micro controllers are well known in industry and are sold at inexpensive prices, both factors in their choice for this project. In addition, there is a large, open-source community supporting them and they were compatible with the programs used in this project. The Arduino Due contains 12 analog input/output ports and 54 digital input/output ports that proved useful for interfacing electrical components to. The analog ports made it possible to get good readings from the accelerometer and the gyroscope.

The digital ports were used to input the readings from the motor encoders and output values to the H bridge in order to drive the motors. Pulse width modulation could also be utilised by the Arduino due in order to control the speed of the motors.

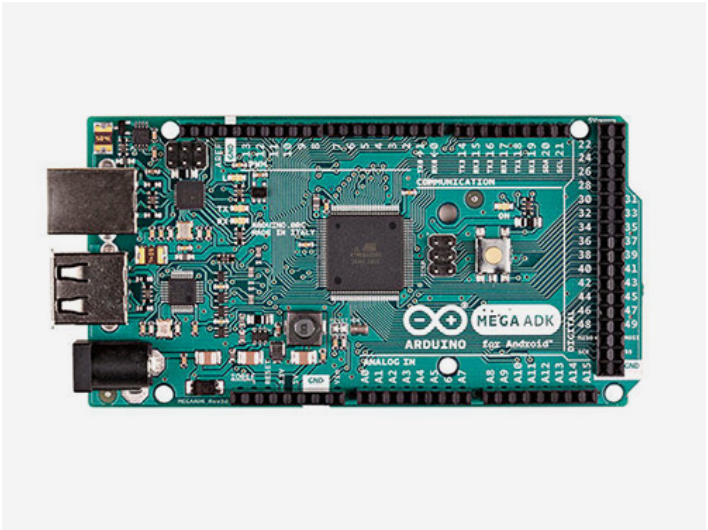


Figure 5 - Arduino Due

3.1.2 Accelerometer (ADXL335 3-axis accelerometer):

The ADXL335 3-axis accelerometer was chosen as it was an effective and inexpensive accelerometer that could offer an accurate angular acceleration data across 3 axis.

The accelerometer measures in m/s^2 (meters per second squared) and has an analog output in volts. The accelerometer will read gravitational effects acting upon it as well as movement. These values will affect the voltage outputted by the gyroscope and this voltage reading can then be used to gain measurements in m/s^2 which can be translated to give a measurement for angle.

The ADXL335 is a capacitive accelerometer, therefore, it was made up of a silicone mass connected to a spring system and hung between two plates of a capacitor. The setup is shown below (see Figure 5):

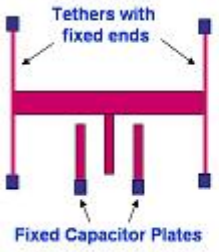


Figure 6 A Differential Capacitive Accelerometer Mass-Spring System at rest (http://eesemi.com/accelerometers.htm) the basic theory states that when acceleration acts on the silicone mass it causes a displacement, shown below (see Figure 7):

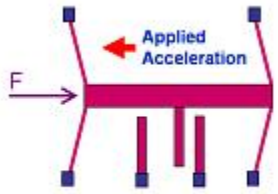


Figure 7 A Differential Capacitive Accelerometer Mass-Spring System subject to acceleration (http://eesemi.com/accelerometers.htm)

This will change the capacitance acting between the silicone mass and each plate and can be measured to give a value for acceleration.

3.1.3 Gyroscope (ENC-03RC Single-axis Gyroscope):

The ENC-03RC Single-axis Gyroscope was also chosen as it has been proved to be effective and inexpensive. The sensor was able to measure the angular rate of one axis, which was more than suitable for the project at hand.

The angular rate measured was measured in degrees per second as shown on the data sheet and contains an analog output for this value.

The angular rate could also be integrated to give an angle of tilt for the platform. This was tested later in the timescale of the project (see 4.2.1 Sensor Readings)

The gyroscope is made up of a mass that is shifted as the gyroscope rotates (see Figure 8 - Internal operational view of a MEMS gyro sensor (https://learn.sparkfun.com/tutorials/gyroscope/all).This movement transmits small electrical currents that can be translated by the circuitry of the gyroscope into a value for angular rate output in volts.

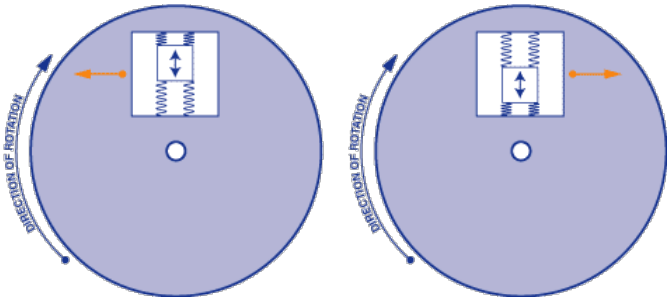


Figure 8 - Internal operational view of a MEMS gyro sensor (https://learn.sparkfun.com/tutorials/gyroscope/all)

3.1.4 Motor (EMG30)

The EMG =30 motor is a commonly used DC-motor that is found in many small to medium projects. The speed of the motor could easily be controlled through the use of PMW and they were found to have a rated speed of 170rpm. The EMG30 comes included with rotary encoders that can be used to measure the position of the wheels and in turn the speed of the motor. As well as this, it features noise suppression through the use of a capacitor across the motor windings. Two motors were used.

These motors were compatible and therefor connected to a Neuftech L298n dual H-bridge. The H-bridge pro was used to control the speed and direction of each individual motor.

The H-Bridge was necessary to supply the DC motors with a voltage and current strong enough to power the motors. This is achieved through the connection of a 7.5Volt lipoly battery to the H-Bridge. The power from the battery was output through two outputs, containing a live connection and a ground, on the H Bridge, one for each motor.

The H-bridge also had a 5v output that was used to power the Arduino board.

3.2 Interfacing the components

The basic model for the system can be described as follows (see Figure 9):

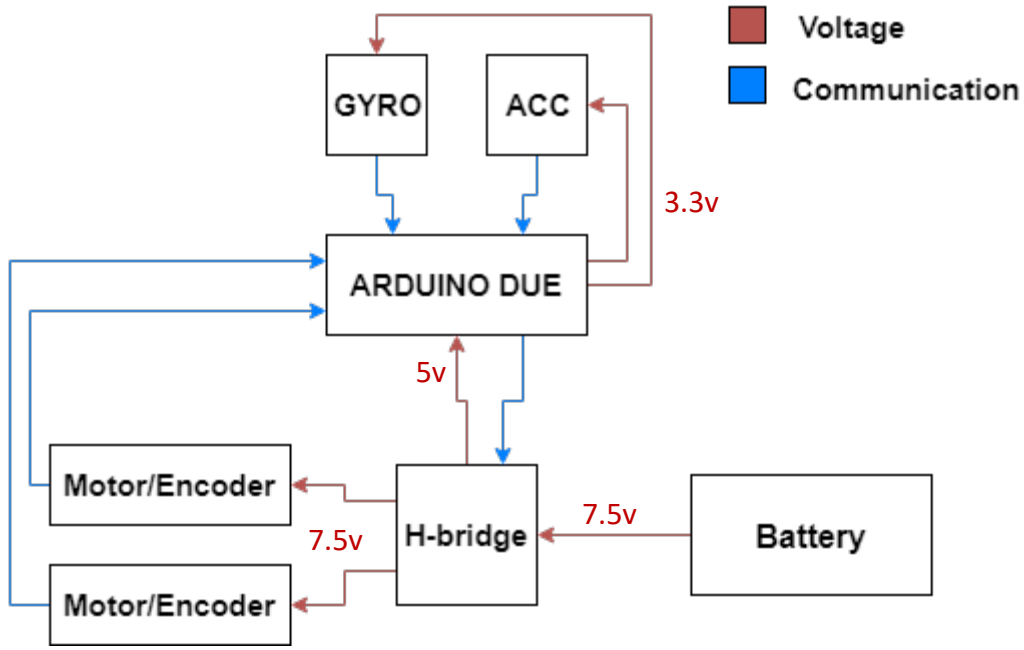


Figure 9

The blue lines represent data being communicated from one device to another and the red lines represent the power supplied to each component. The battery was used to power the H-Bridge. The H-Bridge then was used to output 7.5 volts to power the motors and 5 volts to power the Arduino board. The Arduino board then powered the gyroscope, accelerometer and motor encoders with a 3.3v supply. Pull up resistors were used to ensure that the encoders were either in a high or a low state. The accelerometer, gyroscope and encoder data were fed back to the Arduino board which was used to process the information and control the motors accordingly.

3.3 Creating the Mechanical Structure

3.3.1 Construction of the frame

The design of the frame was an essential part of the project and had to be carefully contemplated otherwise the robot might not balance. Due to this certain design requirements were considered. These were as follows:

- Tall and sturdy design
- Ability to mount electronics and motors
- Ability to drive indoors

As stated earlier in the literature review, a stacked design proved useful for the height and worked well when considering the placement of the electronics.

The robot was expected to run inside on smooth surfaces therefore the EMG30 motor and wheel set was used. This set made use of 100mm diameter wheel with a rubberised tread. The set contained brackets used to mount the motors and wheels. The shelfing for the robot was specifically designed to match the width of the wheel brackets. The length chosen had to be large enough that both motors could be mounted on the underside of the construction. The wheel brackets were measured and then individually modelled on the CAD software. The structure was then created by assembling the parts with screw rods.

A render of the physical model was constructed on CAD software to give an idea of the basic frame of the robot (Figure 10).

Figure 10

A schematic drawing of the basic frame could then be created using the software and this was used to physically build a prototype of the model using sheet plastic and screw rods.
The motors were then attached to the brackets.

The prototyped design did not meet the design requirements, as the frame was not sufficiently sturdy. This was due to the plastic being too flimsy and the robot's width did not provide adequate room to accommodate the motors placed on the underside of the robot. To fix this issue, the robot was rebuilt using 2mm sheet aluminium and the design was widened to accommodate more space for the motors.

The Arduino board, the breadboard, the battery and the H-Bridge were then added to the structure. It was decided to place the breadboard and the Arduino board on the top shelf for ease of access and interfacing components. The battery was found to be the heaviest components. To raise the centre of mass and maintain that it is located above the pivot point the battery was connected to the underside of the top shelf. Lastly the H-Bridge was connected to the middle shelf.

The physical construction was then constructed on CAD software (See Figure 11).
As some of the components were too difficult to construct accurately, they were constructed as boxes on the software. Each component's individual mass was then updated.
It should be noted that wires were not included in the CAD model due to their complexity to model.

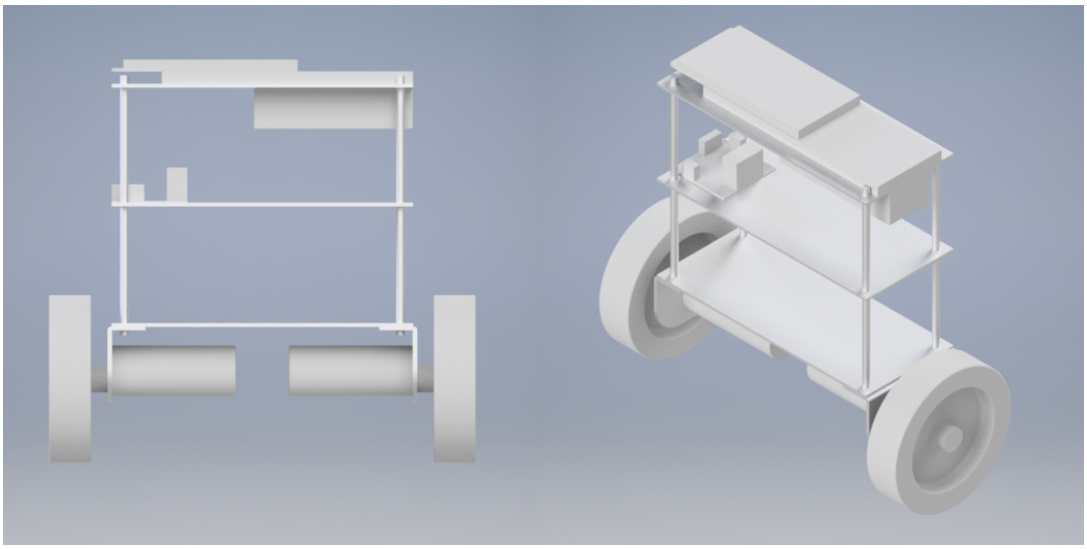


Figure 11

3.3.2 Mechanical Properties

The mechanical properties of the robot were required in order to mathematically model the system further on in the project and were taken as follows:

Total Mass

The robot's mass was measured using scales and the mass was found to be equal to: **1.240kg**

Mass of Pendulum

The pendulum was taken to be the shelfed frame of the robot along with the battery, H-bridge, breadboard and Arduino board. As shown to the right (Figure 12)

The structure was measured using scales and the mass was found to be equal to: ***0.710Kg***

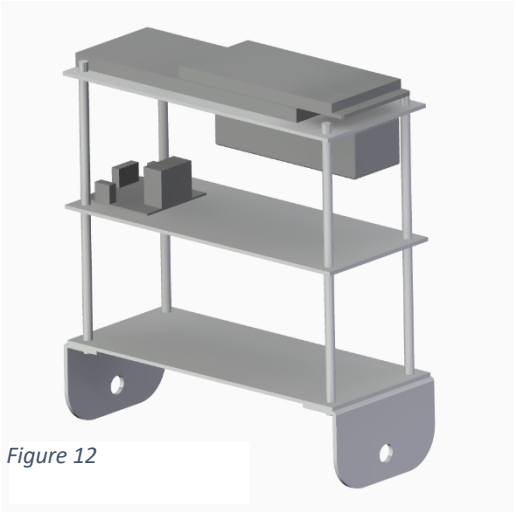


Figure 12

Mass of Cart

The mass of the cart was taken to be the combined mass of the wheels and the motors. As shown to the right (Figure 13)

The structure was measured using scales and the mass was found to be equal to: ***0.530Kg***

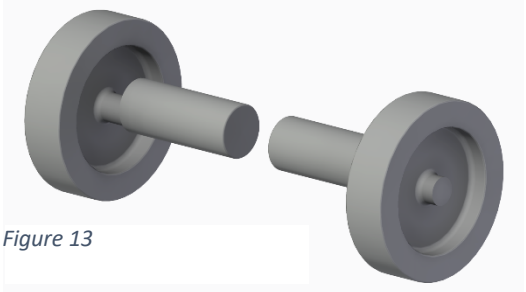


Figure 13

Centre of Mass

The centre of mass was calculated using a method known as the "string method" (Hamid Reza Memarbashi, 2010). Due to the uniform nature of the robot, it is extremely difficult to measure centre of mass using mathematic calculations. Using the string method helped to address these difficulties. This involved suspending the robot by a string from two distinct points. The extended trajectory for each piece of string was then traced through the robot and recorded on paper. The string is expected to pass through the centre of mass and where the pieces of string intersect is where the centre of mass is found. Using this method, the centre of mass was found to be located at **0.140 meters** in the y direction.

Inertia of the Pendulum

To calculate the inertia the robot the metal frame and electronics were modelled together as a block. This was due to the complex nature of the robot's geometry.

The following formula for inertia was used. (Hamid Reza Memarbashi, 2010):

$$I_c = \frac{1}{12} M(h^2 + d^2)$$

This gave a calculated inertia of $\mathbf{2.942\ x\ 10^{-3}\ Kg/m^2}$

Inertia of the Wheels

The inertia of the wheels was also calculated. The wheels were modelled as a cylinder as similarly to the frame of the robot, the geometer was deemed too complex.

The following formula for inertia was used (Hamid Reza Memarbashi, 2010):

$$I_w = \frac{1}{2} M r^2$$

This gave a calculated inertia of $\mathbf{1.751\ x\ 10^{-4}\ Kg/m^2}$ to each wheel

3.3.3 Verification of Mechanical Properties using CAD

In order to verify that the calculations for mechanical properties were correct, CAD software known as Autodesk Inventor was used.

Each part of the robot was created and assembled in CAD software, to give a representation of the physical model. Each part's properties (such as mass) were then edited accordingly. This was to give the model more accurate values for centre of mass and inertia.

It should be noted that some parts were too complex to model and so their geometry was simplified.

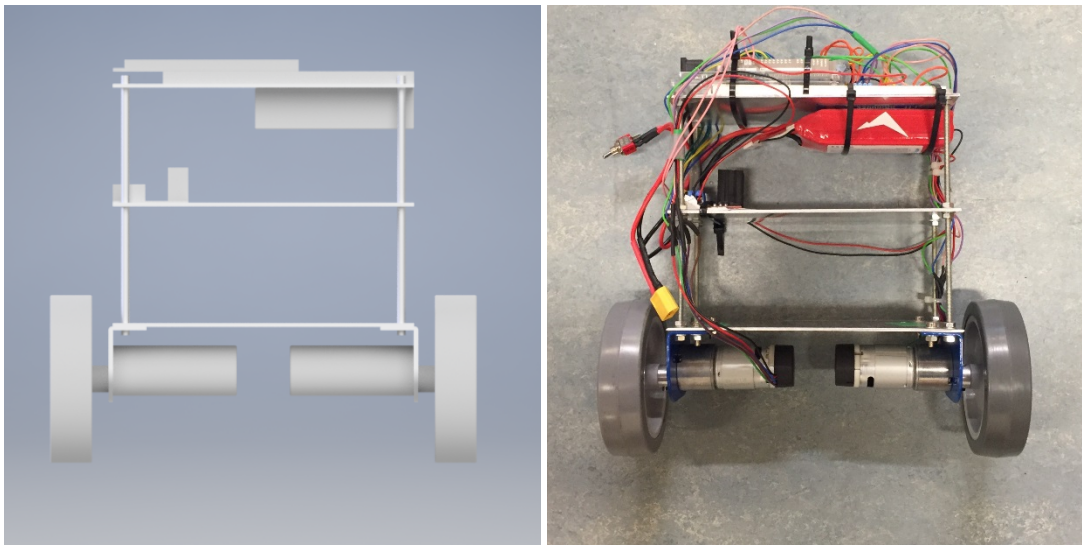
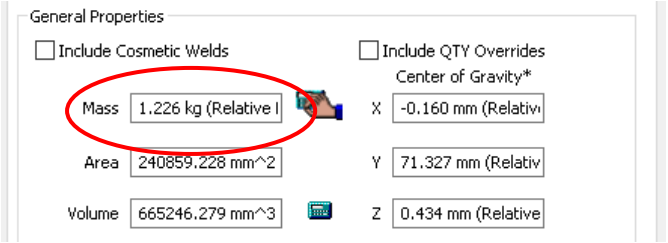


Figure 14 – CAD model (left) picture of real life model (right)

The following comparison can be found above (See Figure 14):

Total Mass

The total mass of the robot was found to be **1.226Kg**. However, this was found to be less accurate than the mass measured by the scales, as the CAD model did not consider the mass of the wiring, electronics, bolts or the added switch. This explained a difference of 14g.



Mass of Pendulum

The total mass of the pendulum was found to be **0.696kg**. The original mass measured using the scales was taken for the same reasoning as the total mass.

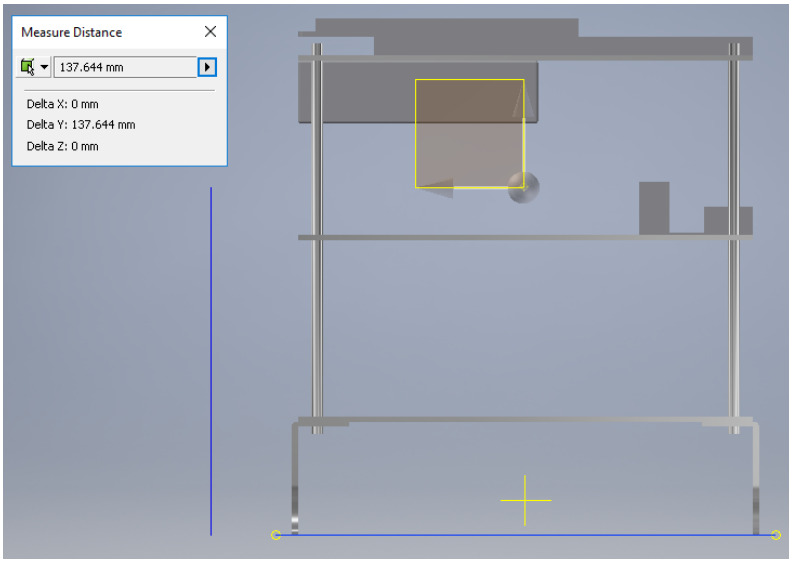
Mass of Cart

The mass of the cart was found to be identical to the mass measured.

Centre of Mass, l

The centre of mass was analysed on CAD and measured to be **0.1376m.**

As the CAD programme's determination was found to be more accurate than the string method the value for COM was taken as **0.1376m.**



Inertia of the Pendulum and Wheels

The inertia of the pendulum and the wheels could not be calculated using CAD software as the correct add on software could not be utilised due to licencing problems.

3.3.4 Summary of values used

A summary of the values used in the rest of the project was constructed as follows:

Property	***Value***	***Unit***
Total Mass of the Robot	***1.240***	***Kg***
Mass of Pendulum	***0.710***	***Kg***
Mass of Cart	***0.530***	***Kg***
Centre of Mass	**0.1376**	***Meters***
Inertia of Pendulum	***2.942 x 10^-3***	***Kg/m^2***
Inertia of wheels	***1.751 x 10^-4***	***Kg/m^2***

3.4 Investigating Simulink and the Arduino Toolbox

3.4.1 Simulink

For this project Matlab's Simulink was used. It was found to be an effective tool as it was compatible with the hardware used throughout the project.

Simulink is a tool commonly used to simulate robotics and control systems. It is a graphical programming system that allows the user to drag and drop blocks from libraries to create a system.

Basic testing was done to become familiar with the software and retrieve results. This mainly was done by reading the values of the sensors that were connected to the Arduino board by running them through a scope.

3.4.2 Arduino Toolbox for Simulink

The Arduino toolbox is an add-on toolbox for Simulink that allows the user to create and run programs for an Arduino board on Simulink. This allows the user to make use of input and output blocks to receive and output values from the input and output ports of the Arduino board.

3.5 Mathematical Modelling of the Robot

3.5.1 Introduction to Mathematical Modelling

Mathematically modelling the self-balancing robot proved to be a very difficult task. The system is highly non-linear which makes it very complex. Therefore, techniques to linearize the system had to be investigated. To mathematically model the system, the transfer functions for each part of the system were determined.

3.5.2 Basic explanation of a closed loop system

A closed loop system will contain an input signal and an output signal. The input signal is also known as the desired input and is set at a value that is desired from the output. In this case, the desired input is 0°. The output signal is simply the output of the system. The output signal of a closed loop system will constantly be monitored by a transducer in the feedback loop– in this case by the motor encoders in one loop and the gyroscope and accelerometer in the other. This will then be fed back through a summing junction to the input signal. The summing junction subtracts the value from the desired angle to give bring the actual angle closer to the value of 0. The error of the system is simply the difference between the input and the output.

From this knowledge an example block diagram of a closed loop system was drawn out as follows (see Figure 14).

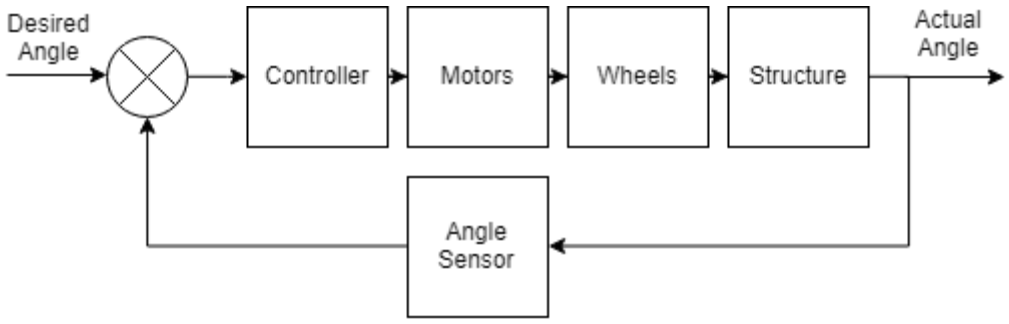


Figure 15

3.5.3 Design of the robots closed loop system

The robots closed loop program was designed to keep the desired angle for the robot at 0 as it was desired to be kept in the upright position.
The system was designed as a closed loop system with two feedback loops - a speed controller loop and an angular control loop.

The two loops used in this system were designed to keep the angle and speed of the motors as close to zero as possible. As the system is highly unstable the robot will fall easily with little or no external force.
Modelling the system will give a good idea of how the system works and what values could be used to implementing a PID controller to the system.
In order to get a good understanding of the system, the following block diagram was created (Figure 16).

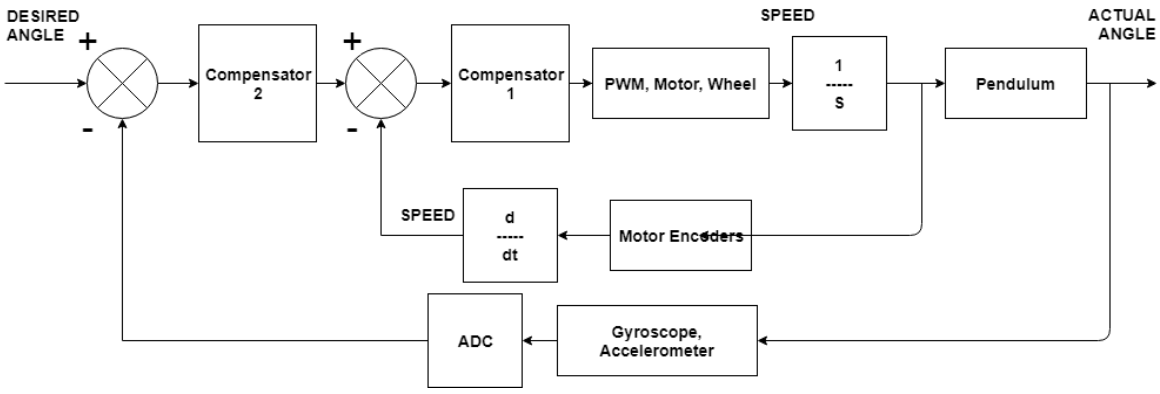


Figure 16

3.5.4 Transfer functions of the speed controller loop

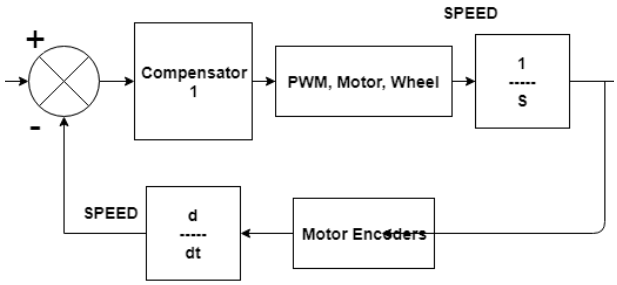


Figure 17

To create and simulate the speed controller loop, transfer functions for the pulse width modulator, the motor and the wheel had to be modelled.

For this the transfer functions for the PWM, motor and wheel were taken from a past project and given by the project supervisor.

The transfer function was given as:

$$\frac{14}{1 + 0.2s}$$

The step input for the motors transfer function can be shown below (see Figure 18):

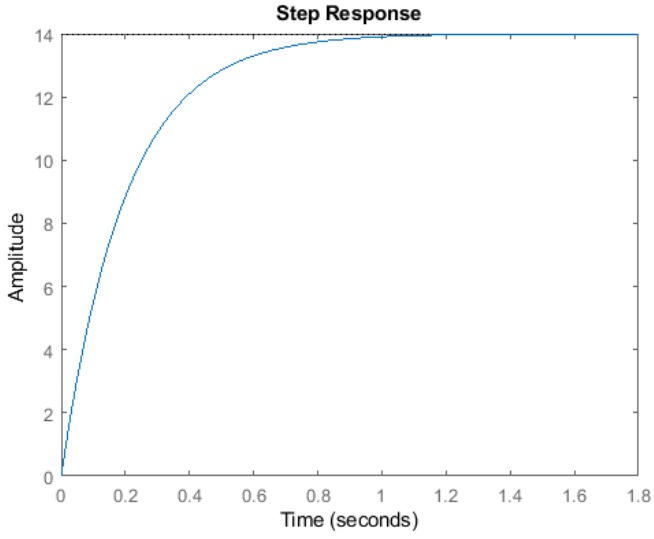


Figure 18

The unit used by the motor encoders was given as 1 bit per radian. As this was read when reading an output from the encoders through a scope in the system.
This gave the following:

$$\frac{1}{0.0174533}$$

The desired input for angular rate was fed through the system and then integrated to give a value for angular position. This was fed back through the feedback loop and through the transfer function for the motor encoders. This signal was then differentiated to get a value back in angular rate to feed back into the summing junction.

The flowing loops were run on Simulink:

An impulse of 1 bit was input into both loops shown below:

The first loop contained the transfer function, previously discussed for the motor. This was integrated to give a value for position.
The feedback loop contained the unit for the encoders which was in radians per second. This was differentiated to give angular rate which was fed back into the summing junction.

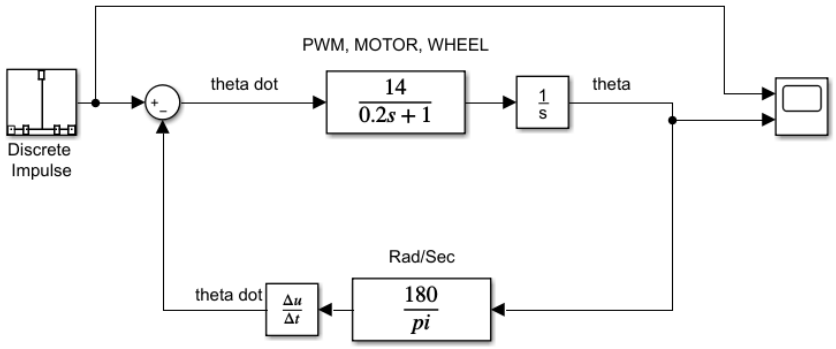


Figure 19

The results from this test are show below and show that an impulse of 1 bit gives a value of 0.017radians – the equivalent of 1 degree – which shows the wheels position will move by 1 degree (see Figure 20).

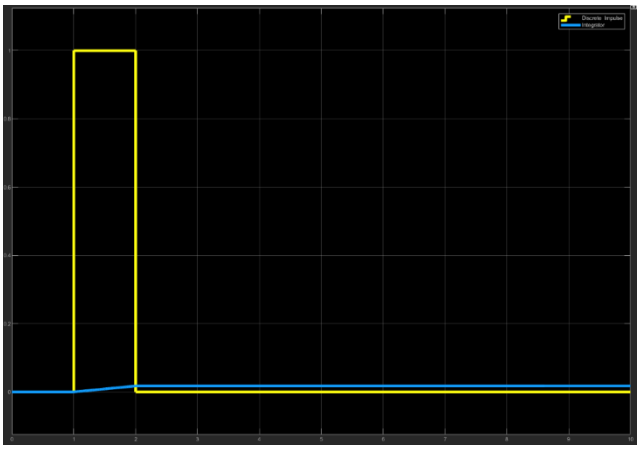


Figure 20

The second test produces angular rate, as the integration involved is removed.

As shown below (Figure 21) the impulse causes the angular rate to jump up to a speed of 0.017radians/second – the equivalent of 1 degree/second – which shows the wheel speed will move at 1 degree/second.

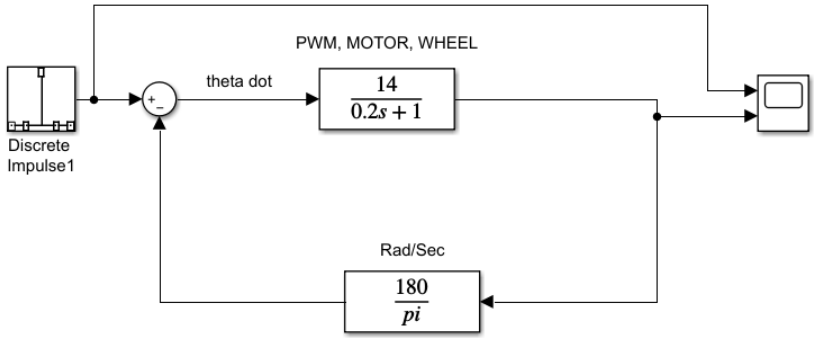
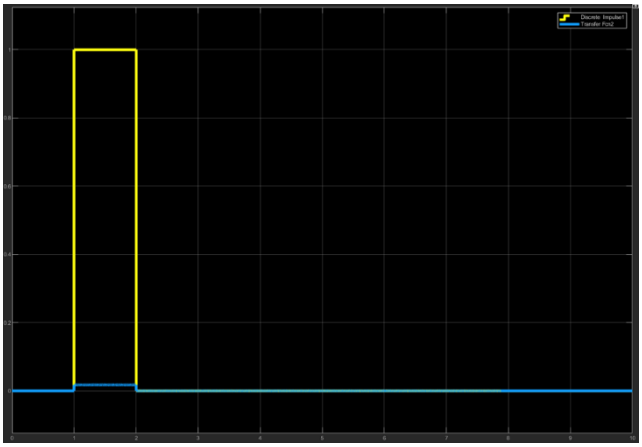


Figure 21



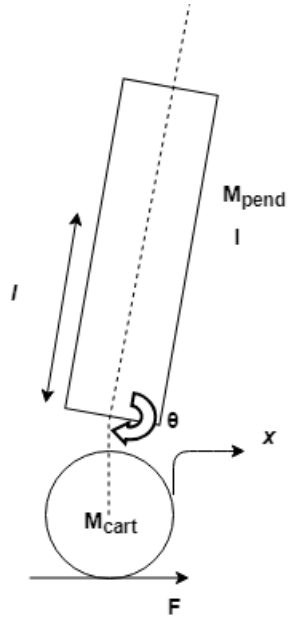
3.5.5 Transfer functions of the Angular loop

The speed controller loop produced a value for the position of the wheels.
The desired output for the system was the position of the pendulum in radians. To achieve this a mathematical model of the robot had to be found.

The mathematical model calculated gave a transfer function between input $U(S)$ and output φ(S).

Calculating the Transfer Function of the Pendulum

The system was modelled after an inverted pendulum, constrained in the sagittal plane. The theory and MATLAB code for this section was heavily based off the MATLAB Inverted Pendulum: System Modelling tutorial (MATLAB® 9.2).
In the inverted pendulum the controlling input is the force applied to move the cart, whereas in this system, the controlling input was the torque applied from the motors to the wheels, which affected the horizontal force applied to the system. (see Figure 22).



Mpend = Mass Pendulum

Mcart = Mass of Cart

I = Mass moment of inertia

l = Length to centre of gravity

Theta = Pendulum angle from vertical

F = Force applied to wheels

x = Position of the wheels

Figure 22

Force Analysis and Equations of Motion

The free body diagrams for the pendulum and the cart had to be drawn and analysed. (see Figure 23). The free body diagram shows the forces that are acting on the system where the forces acting on the pendulum are in red and the forces acting on the cart are in blue.

Forces P and N represent the forces acting on each body from the opposite body.

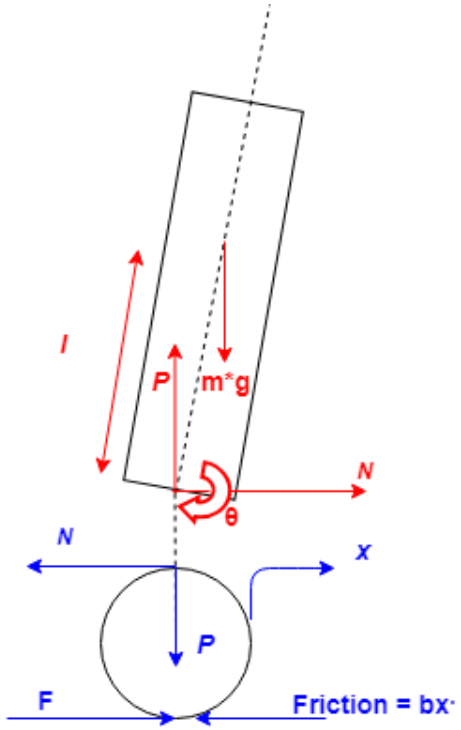


Figure 23

Summing the forces for the cart in the horizontal direction will give the following equation of motion:

$$M_{cart}\ddot{x} = F - N - b\dot{x}$$

It is rearranged to give the following:

$$F = M_{cart}\ddot{x} + b\dot{x} + N$$

Friction was then eliminated from the equation, as a value could not be determined. Therefore the equation changed as follows:

$$\boldsymbol{F = M_{cart}\ddot{x} + N}$$

The forces for the pendulum were then summed in the horizontal direction, to give the following equation:

$$M_{pend}\ddot{x} = M_{pend}l\dot{\theta}^2 * Sin\theta - M_{pend}l\ddot{\theta} * Cos\theta + N$$

It was rearranged to give the following:

$$\boldsymbol{N = M_{pend}\ddot{x} + M_{pend}l\ddot{\theta} * Cos\theta - M_{pend}l\dot{\theta}^2 * Sin\theta}$$

These equations could then be substituted into each other, to give the first of two equations of motion needed for the system:

$$\boxed{\boldsymbol{F = (M_{cart} + M_{pend})\ddot{x} + M_{pend}l\ddot{\theta} * Cos\theta - M_{pend}l\dot{\theta}^2 * Sin\theta}}$$

To get the second equation, firstly the equations of motion had to be summed, perpendicular to the pendulum. This gave the following equation:

$$P * sin\theta + N * \cos\theta - M_{pend}g * sin\theta = M_{pend}l\ddot{\theta} + M_{pend}\ddot{x} * \cos\theta$$

The moments were then summed to give the following equation:

$$-Pl * \sin\theta - Nl * \cos\theta = I\,\ddot{\theta}$$

Combining these 2 equations gave the second of two equations of motion:

$$\boldsymbol{\left(I + M_{pend}l^2\right)\ddot{\theta} + M_{pend}lg\sin\theta = M_{pend}l\ddot{x}\cos\theta}$$

As these equations are not linear in respect to θ the equation had to be linearized.

Linearizing the system.

As stated earlier the system was a highly non-linear one and to control the robot, the system was linearized about the vertical axis. This was done by using small angle approximation. This is technique used to simplify trigonometric functions.

At the robot's upright position

$\boldsymbol{\theta = 180}$

$\boldsymbol{\varphi = 0}$

φ was taken to be the angle between the pendulum and the vertical axis.

From the graph below (see Figure 25) it was shown that **at small angles**, φ was approximately equal to $\sin\varphi$ and $1 - \cos(\varphi)$**.** Therefore, we can state that at small angles:

$$cos\theta = -\cos(\theta + \varphi) \approx -1$$
$$sin\theta = -\sin(\theta + \varphi) \approx -\varphi$$
$$\dot{\theta}^2 = \dot{\varphi}^2 \approx 0$$

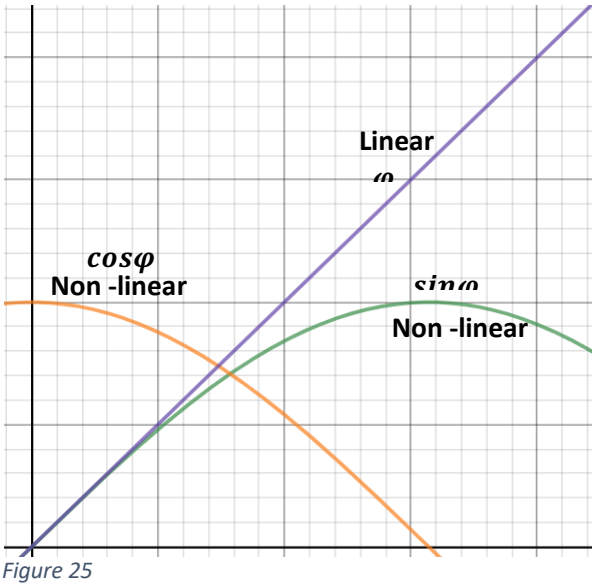


Figure 25

These values for $cos\theta$ and $sin\theta$ were then substituted into the equations to give linear equations of motion.

$$\boldsymbol{F = (M_{cart} + M_{pend})\ddot{x} - M_{pend}l\ddot{\theta}}$$

$$\boldsymbol{(I + M_{pend}l^2)\ddot{\varphi} - M_{pend}lg\varphi = -M_{pend}l\ddot{x}}$$

Transfer function of the Cart and Pendulum

To create the system's transfer functions, the Laplace transform of both equations of motion had to be obtained.

These are shown below:

$\boldsymbol{F = (M_{cart} + M_{pend})\ddot{x} - M_{pend}l\ddot{\theta}}$ $\qquad$ $\boldsymbol{(I + M_{pend}l^2)\ddot{\varphi} - M_{pend}lg\dot{\varphi} = -M_{pend}l\ddot{x}}$

Equation 1 $\qquad$ Equation 2

$\boldsymbol{F = (M_{cart} + M_{pend})X(s)s^2 - M_{pend}l\Phi(s)s^2}$ $\qquad$ $\boldsymbol{(I + M_{pend}l^2)\Phi(s)s^2 - M_{pend}lg\Phi(s) = -M_{pend}lX(s)s^2}$

Equation 1 was then rearranged to get $\boldsymbol{X(s)}$ where:

$$X(s) = \left(\frac{I + M_{pend}l^2}{M_{pend}l} - \frac{g}{s^2}\right)\Phi(s)$$

This was then substituted into equation 2 to get the following equation:

$$U(s) = \left(M_{pend} + M_{cart}\right)\left(\frac{I + M_{cart}l^2}{M_{cart}l} - \frac{g}{s^2}\right)\Phi(s)s^2 - M_{pend}lg\Phi(s)s^2$$

Subsequently, this was rearranged to get the following transfer functions for pendulum position. The second transfer function for cart position was derived as follows:

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{M_{pend} * l * s}{q}}{s^3 - \frac{\left(\left(M_{pend} + M_{cart}\right) * M_{cart} * g * l\right) * s}{q}}$$

A value labelled 'q' was added to allow for easier coding and was described as follows:

q $\quad = (M_{cart} + M_{pend}) * (I + m * l\text{^}2) - (m * l)\text{^}2$

The following properties required for the transfer function were calculated. These were found to be the following values:

Mpend = 0.710 Kg

Mcart = 0.530 Kg

I = 2.942*$10^{-3} Kg/m^2$

l = 0.1367m

Substituting these values into the equation gave the transfer function below (see Figure 26 – Transfer function for the pendulum).

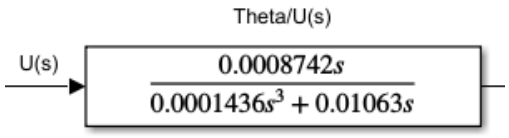


Figure 26 – Transfer function for the pendulum

Converting Angle of the Wheel to Force

The transfer function for the pendulum was given as $\frac{\varphi(s)}{U(s)}$. Therefore, to receive the angle of the pendulum, U(s) was required as an input.

This can be acquired by deriving a transfer function from the following equation:

$$F + r = I_{wheel}\,\omega^2$$
$$F + r = I_{wheel}\,\ddot{\theta}_{wheel}$$

$$F = \frac{I_{wheel}\,\ddot{\theta}_{wheel}}{r}$$

$$F = \frac{I_{wheel}\,S^2\,\theta_{wheel}}{r}$$

$$\frac{U(S)}{\theta_{wheel}(S)} = \frac{I_{wheel}\,S^2}{r}$$

The values for the parameters were then substituted into the equation to give the following transfer function:

$$\frac{U(S)}{\theta(s)} = \frac{0.000175125\,S^2}{0.0005}$$

This was represented as the following on Simulink:

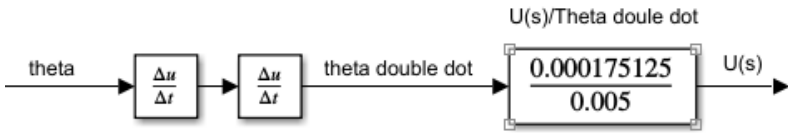


Figure 27

Feedback loop

The feedback loop contains the transfer values for the accelerometer blocks and the analog to digital converters.

The signal was split and the signal going to the gyroscope was differentiated to convert φ to $\dot{\varphi}$, as this was measured by the gyroscope.

The sensitivity of each device was then inserted, to convert φ to volts.

The ADC converted the analog voltage to a digital value measured in bits. The Arduino board supplied the sensors with 3.3v and the ADC converted this to a scale of 1024. A value of $\frac{1024}{3.3}$ was obtained. This is shown below:

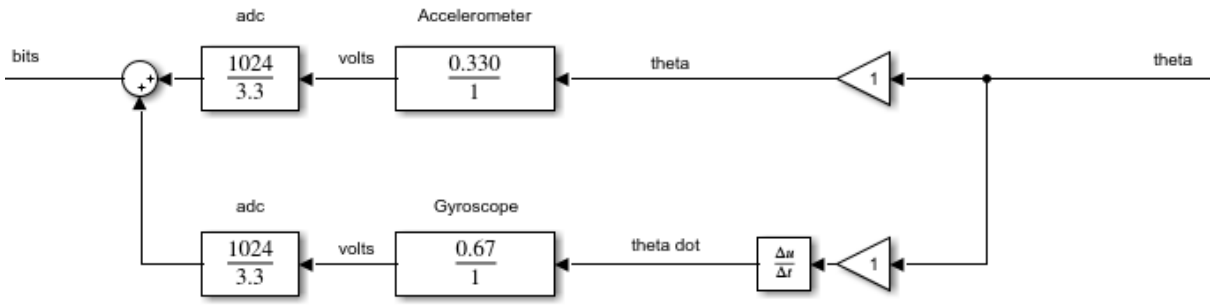


Figure 28

This signal was fed back into a summing junction as an error, which was subtracted from the error signal, to attempt to bring the system back to 0.

This entire system can be found in Appendix 1.1 - Full closed loop Simulink Model

3.5.6 Setting up the Proportional Derivative Algorithm

3.5.6.1 Understanding a PID controller

A PID controller is made up of three controllers:

- Proportional
- Integral
- Differential

Proportional Controller P

The P controller in a PID is known as amplifier gain and can be used to enhance the performance of a closed loop system.
When the amplifier gain in the system is low, the "rise time" (the time the system takes to react) will be very long and the desired output will not be reached.
When the amplifier gain is increased, the rise time will shorten however the output will either oscillate slightly then settle, or start to oscillate continuously.
It should be noted that if the gain is too high, the system will go out of control.

Integral Controller I

The integral controller of a system is usually used in conjunction with a P controller. The integral controller is used to eliminate error i.e. bring the error to 0.

Although the integral controller is effective at eliminating error, in doing so it slows the system down.

Differential controller D

The differential controller is used to increase the rise time of the system and is often only used in conjunction with a proportional or a PI controller. The output of a differential is a pulse which occurs at the instant a step input is applied.

3.5.6.2 The Algorithm

A PD controller is a popular type of controller used in the balancing of a platform using an accelerometer and a gyroscope. (Shane Colton, 2007)

The theory states that a proportional derivative controller can be used to stabilise the system in the form of:

$$System\ output = K_p * angle + K_d * angular\ rate$$

This theory allows the values of K_p and K_d to be altered to stabilise the system.

The theory states that "the PD control scheme is like adding an adjustable spring and damper to the robot"

The systems PD controller was found to have already been set up in the feedback loop, where gain blocks were used to multiply the values of angle and angular rate thereby effectively stabilising the system.

3.6 Using Simulink to Program the Physical Model

The Simulink model used to control the robot was comprised of 3 main sections:

- The motor Controllers
- The Sensor unit
- The Motor Encoders

3.6.1Motor Controller

Driving the Motors

Two "pulse width modulation" blocks were placed as drivers for the motor. (See Figure 29) The blocks received a value of 0 to 255, where 255 outputs the top speed of 170RPM - for the motor equipped to the robot.

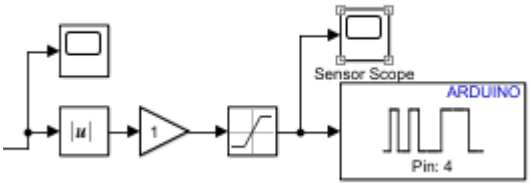


Figure 29

It was found that the pulse width modulation blocks would only receive positive inputs, therefore, a modulus block was inserted, to convert all negative inputs to positive inputs.

Controlling the Direction

The directional control was controlled through the H Bridge.

For each motor there were 2 inputs to control the direction of the motor, IN1 and IN2.

If:

IN1 > IN2 – the motor moves clockwise

IN2 > IN1 – the motor moves anti-clockwise

To control the direction, the signal used to drive the motors was put through two 'compare to zero' blocks.

If the signal was positive, the 'compare to zero' block marked ">0" (greater than zero) would output a binary 1 and the block labelled "<0" would output 0, driving the two motors clockwise. The opposite could be said where the signal was negative.

This is shown to the left (see Figure 31) where a signal has changed from negative to positive. The direction has changed from anti-clockwise to clockwise

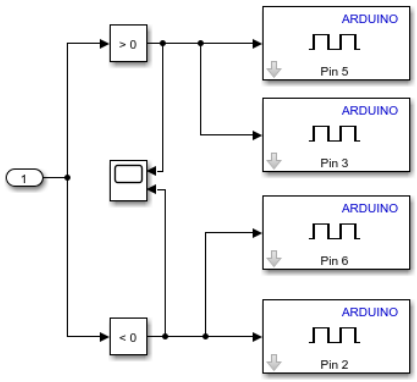


Figure 30

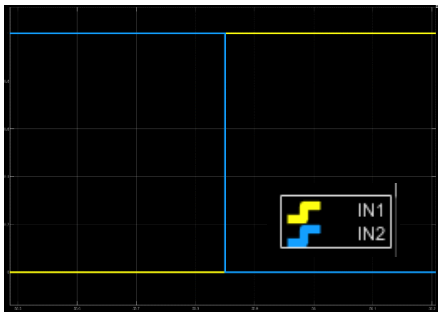


Figure 31

3.6.2 Gyroscope and Accelerometer

The gyroscope and accelerometer were put together to into a subsystem labelled "SENSOR SYSTEM". The input from the Arduino block was sent through an analog to digital converter (known as an ADC). This converted the input voltage of the appropriate sensor from volts to a number of bits between 0 and 1023

Using the gyroscope and Accelerometer to create a Proportional Derivative Controller

To create an effective controller for the self-balancing robot it was decided that both the angular velocity and the angle of the robot were required. These two readings can be used to create the basis of a PD controller.

The accelerometer acted like a proportional controller as when the robot started to fall over the signal to be sent to the motor would start to rise. This can be shown below:

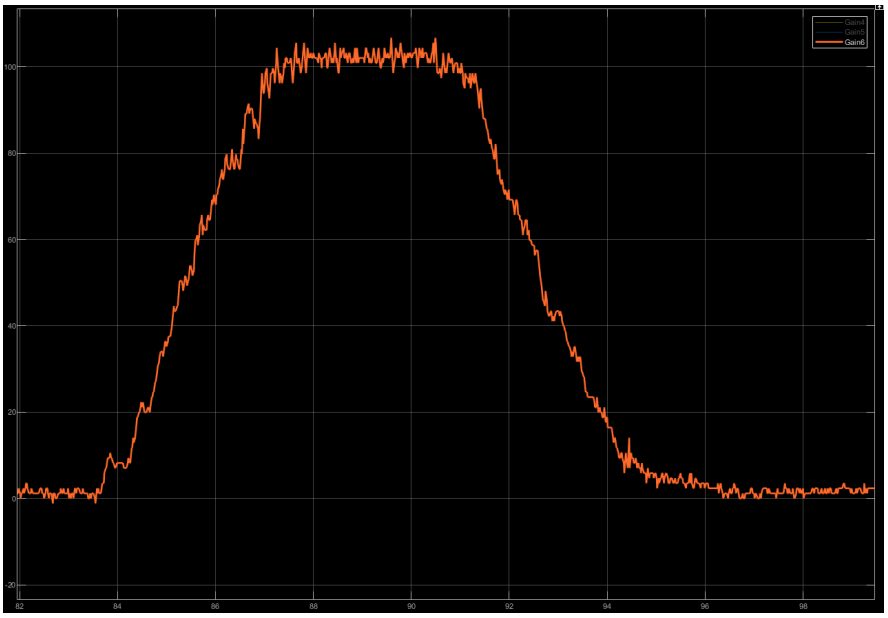


Figure 32 - Accelerometer reading

The gyroscope, when used in conjunction with the accelerometer sped up the system, giving the reading a kick as shown below:

Figure 33 - Combined gyroscope and accelerometer data

Sensor Calibration

The input of both sensors in this case was 3.3 volts and so both inputs were multiplied by $\frac{3.3Volts}{1023Bits}$ to get the reading of the sensors in volts.

This was important when referring back to the data sheets for each component, as the values for sensitivity and the offset are in terms of voltage.

Calibration of the Accelerometer

It proved beneficial to have an accurate reading for angle of rotation. Using the y axis in conjunction with the x axis it was possible to use trigonometry to give an accurate reading for angle. This was done through finding the inverse tangent of the two axis readings.

A simpler method of finding the angle was to use "small angle approximation", using only the x axis. This is an effective method between the angles of 20°

This is where:

x axis = 1g * sin (θ)

Using small angle approximation we can state that:

sin(θ) ≈ θ

This is explained further below.

The reading from the accelerometer was desired in g (gravitational acceleration). To do this, the voltage reading from the gyroscope was passed through a summing junction, combined with a value for the offset, so that at 0° was = 0V out.

The stated offset for the accelerometer on the data sheet was 1.6V. This was tested, and it was found that the value was around 1.629V. As this was approximately 1.6V the reading was accepted (Figure 34 - Signal without offset (yellow), signal without offset (blue)).

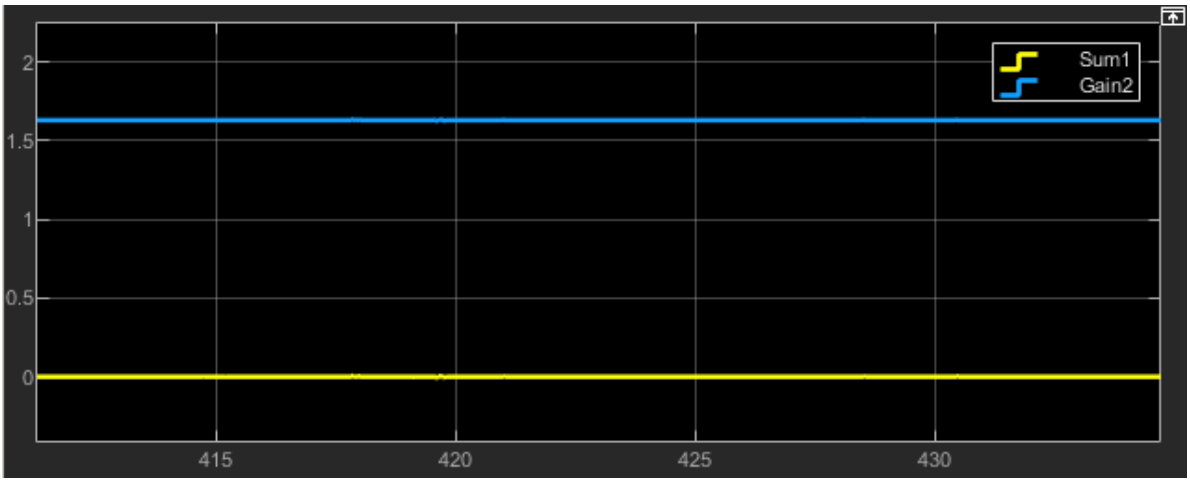


Figure 34 - Signal without offset (yellow), signal without offset (blue)

The sensitivity of the accelerometer was stated to be 0.330V/g. Therefore, the value coming from the sensor was then multiplied by $\frac{1}{0.330}$ to give a reading in g.

This value could be converted to $\mathrm{m/s^2}$ by multiplying through by the gravitational constant of 9.80665 $\mathrm{m/s^2}$. However, this wasn't considered necessary.

At 0°, the accelerometer read 0g as the x axis lies flat. At 90°, the accelerometer read 1g, as the x axis was vertical.

This was proved to be the case when run in the Simulink model (see Figure 35 - accelerometer reading (left), robot at 90° (right)).

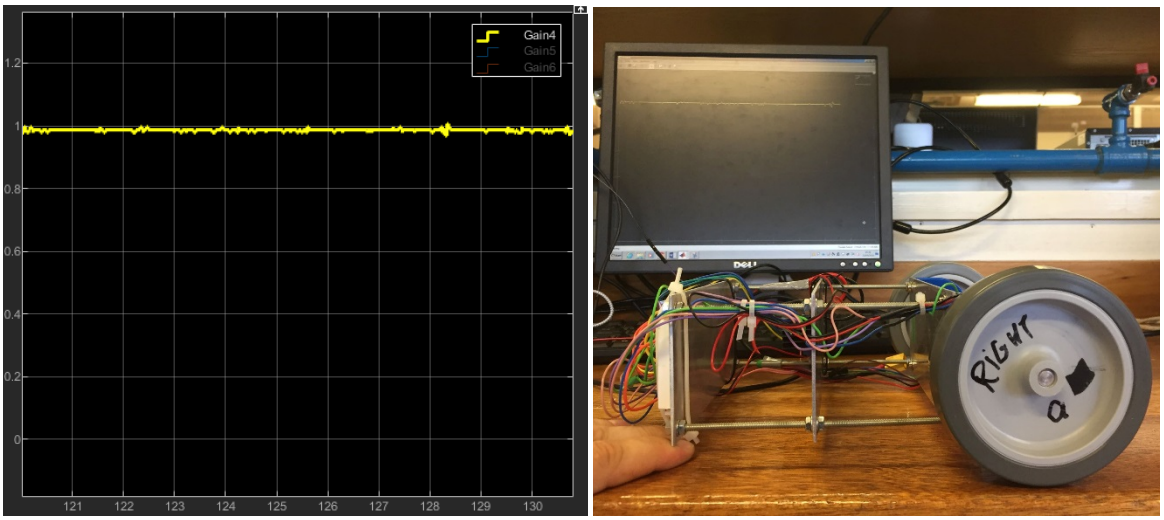


Figure 35 - accelerometer reading (left), robot at 90° (right)

Using small angle approximation, the reading of the accelerometer in g was multiplied through by 90° to get an accurate reading for the angle in degrees.

This was then converted to radians by multiplying the value by $\frac{\pi}{180}$.

Calibration of the Gyroscope

The gyroscope was implemented to provide a value for angular rate. The angle of the robot could also be obtained from the gyroscope through integration of the angular rate value. This was discussed further in section 4.2.1 Sensor Readings

The reading given from the gyroscope was passed through a summing junction, combined with a value for an offset. This was combined so that when the robot was stationary, the value given read 0V. The data sheet stated the offset to be 1.35V. This was tested, and it was found that the value of 1.3226 was more suitable (see Figure 36 - Signal without offset (yellow), signal without offset (blue)). This value was accepted.

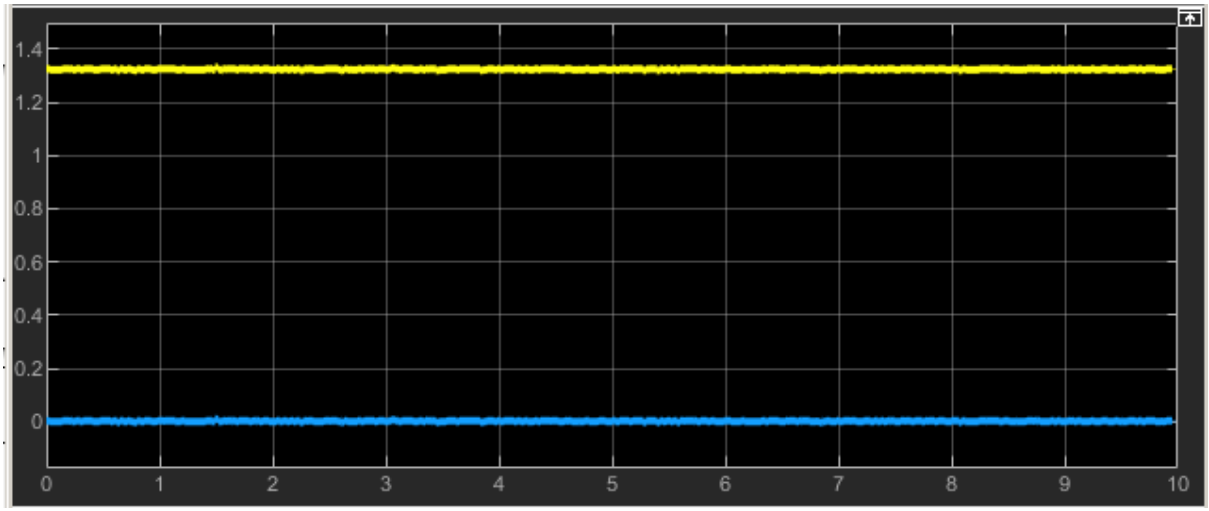


Figure 36 - Signal without offset (yellow), signal without offset (blue)

The sensitivity of the gyroscope was given as 0.67V/degree/second. Therefore, the value coming from the sensor was then multiplied by $\frac{1}{0.67}$ to give a reading in degrees/second.

3.6.3 Motor Encoders

The motor encoders were used to control the speed of the system.

This was used to ensure the speed of each motor was accurate and that both motors were spinning at equal speeds.

Each motor contained 2 encoders. These encoders would read the opposite of each other. Each encoder would either read 0 or 1.

These signals were then placed through an encoder block that was taken from a previous project. The encoder block counted each change in the encoders and was programmed to count 1 bit up for every degree the wheel was turned clockwise and down for every degree the wheel was turned anticlockwise. From this reading, it was possible to read the position of the wheels x.

Once the wheels positional data was available, it was then possible to acquire the wheel speed through the use of a PID block. The PID block was used to differentiate the positional data x to give a value for wheel speed $\dot{x}$.

The wheel speed was then passed through a gain block and fed through a summing junction, where the wheel speed was then subtracted from the angular data. This combined signal was then fed to the motors.

This meant as the angle increased and subsequently the speed of the motors increased, the reading from the encoders would increase and then be subtracted from the signal, slowing the motors down as the angle was corrected towards 0°.

Values for the gain block were estimated, then tested and later calculated through mathematical modelling.

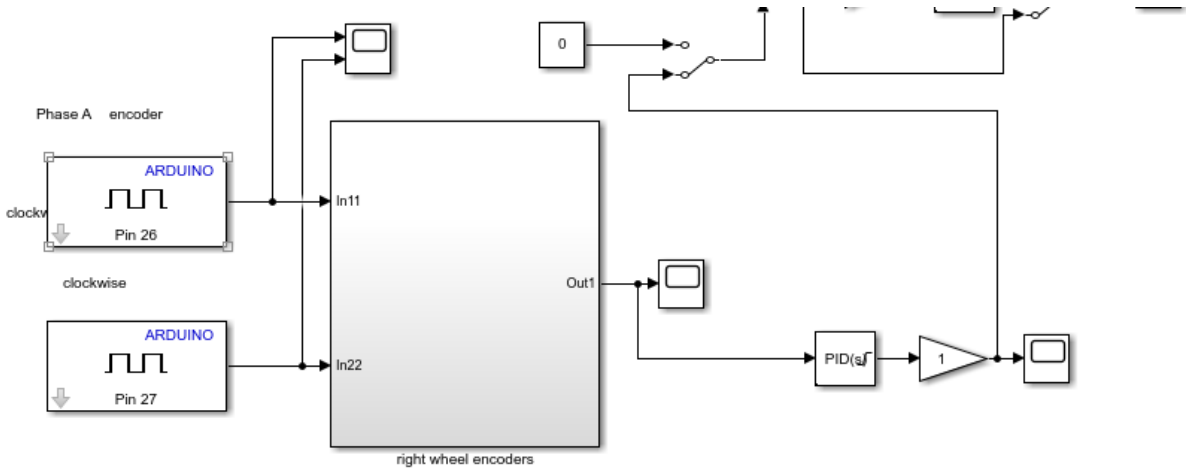


Figure 37

4. Results

4.1 Results from Simulink model

To test the system a discreet impulse response was set up to simulate a disturbance or knock to the robot. This impulse was set to occur 2 seconds into the simulation for a length of 1 second. The whole simulation was set to run for 50 seconds.

Firstly, the closed loop system was tested with no modifications.

The results showed that the angle of the robot would increase dramatically and rapidly oscillate. The

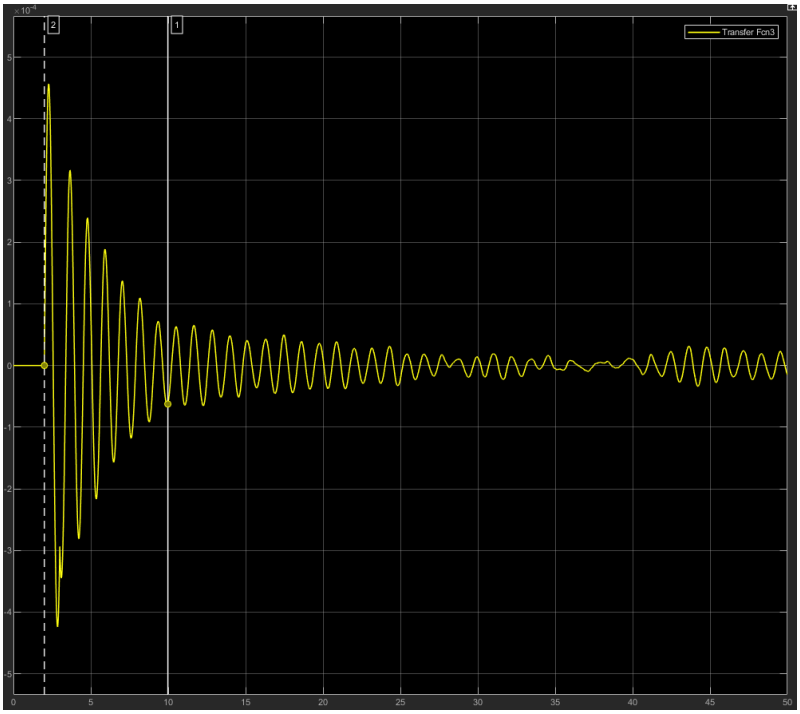


Figure 38

magnitude of the oscillations would gradually decrease but would not settle. They were shown to oscillate at a near constant amplitude and frequency, showing that the system was marginally stable.

It was noted that the magnitude of the scale was not correct. The scale showed that the robot oscillated at less than 1 degree and therefore, it was not known to what extent the robot oscillated. The position and speed of the wheel were also observed.

The graph below (see Figure 39) shows that the position of the wheel would start to increase, then oscillate. It was shown that the wheel position did not return to its original position. The oscillations in position and speed are explained by the movement of the wheels, to correct the robot's angular position.

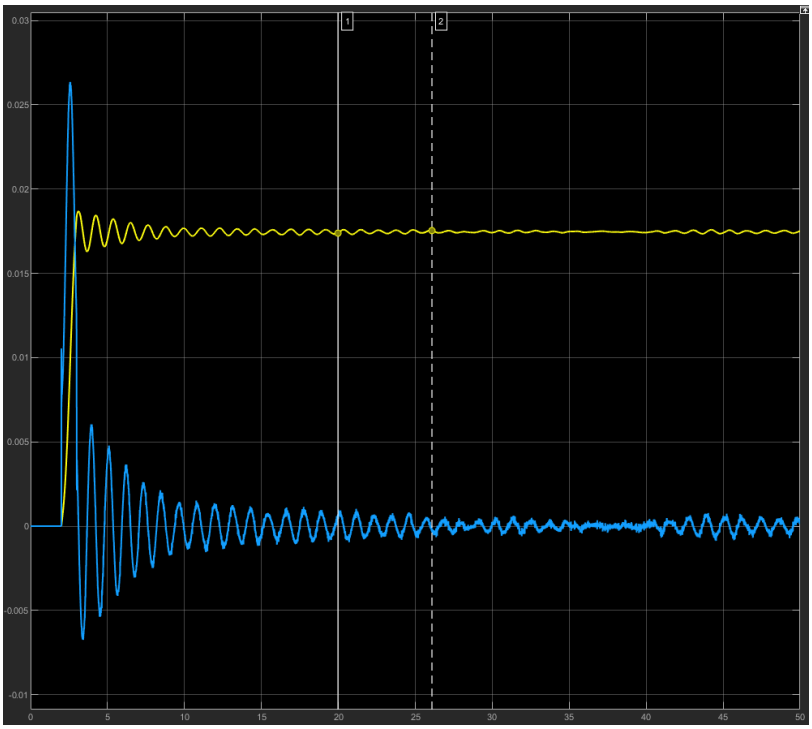


Figure 39 - value for position (yellow), value for speed (blue)

Tuning the PD values.

As stated earlier in section 3.5.6.2 The Algorithm 3.5.6.2 The Algorithm

$$System\ output = K_p * angle + K_d * angular\ rate$$

Where the proportional value was represented by the gain multiplied by the angle and the derivative by the gain multiplied by the angular rate, the gain blocks in the feedback loop that amplify the signals going through the accelerometer and gyroscope data act as a proportional, derivative controller.

To begin, the proportional value was tuned. This was done by continuously adding 1 to the gain block. This was found to settle the system, making it somewhat stable. The more value that was added to the gain block, the faster the settling time would be.

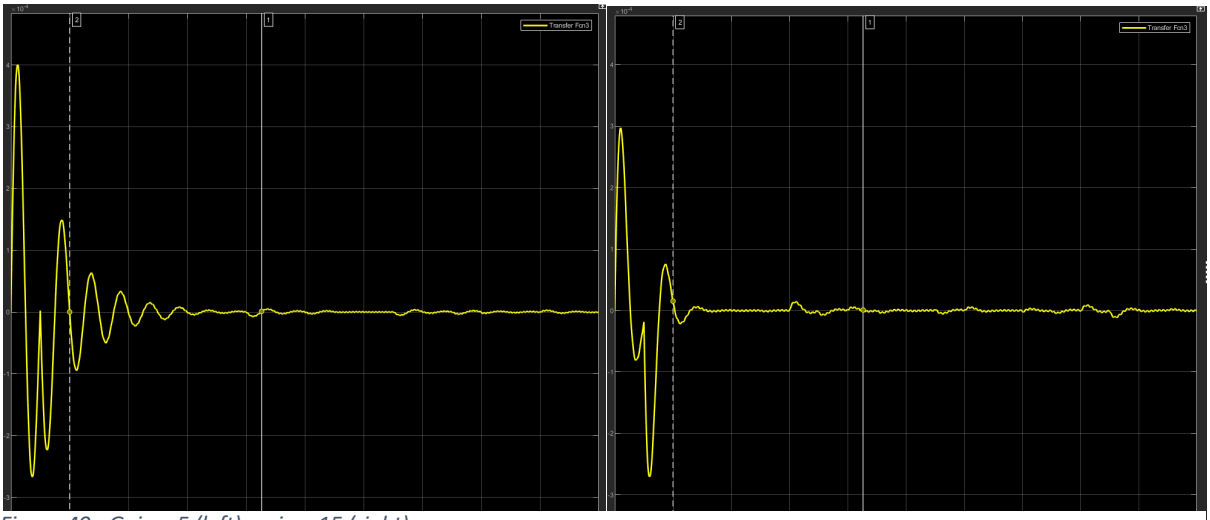


Figure 40 - Gain = 5 (left), gain = 15 (right)

This was done until the value of 15, where it was found that as the gain was further increased, the system would start to oscillate more.

Following this, the differential value was adjusted to heighten and speed up the response of the system, as well as eliminate the oscillations introduced to the system. The value of the gain block was increased from 1.

As the gain blocks value was increased, the resulting output showed a smoother curve and fewer oscillations.

The value was tuned to 4 which gave a settling time of 8 seconds and lowered the response time. Therefore 4 was deemed to give the best results.

With PD tuning, the response of the system was dramatically increased (see Figure 41). Although the system was found to be stable, the settling time of 8 seconds was deemed too long.

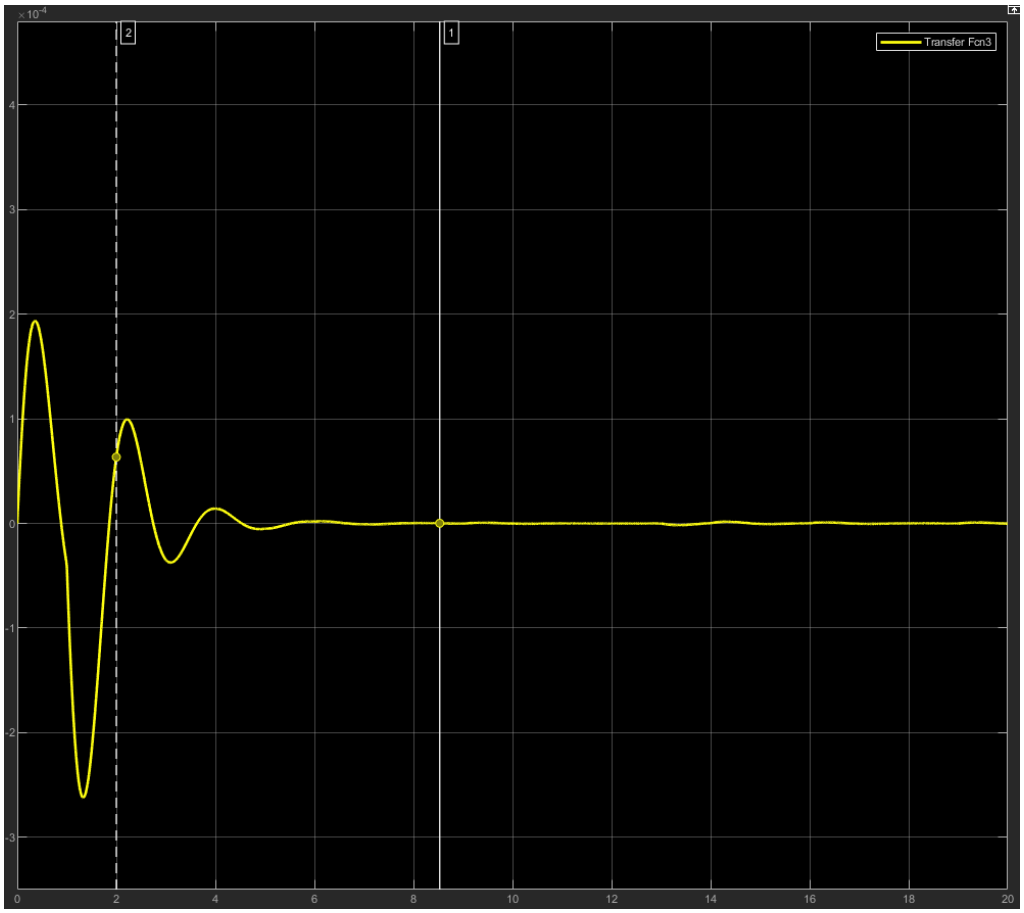


Figure 41 – Final response after PD tuning

4.2 Results from Physical model

4.2.1 Sensor Readings

Angular velocity was acquired by simply taking the calibrated reading from the gyroscope. This reading gave a fast and accurate reading for angular velocity.

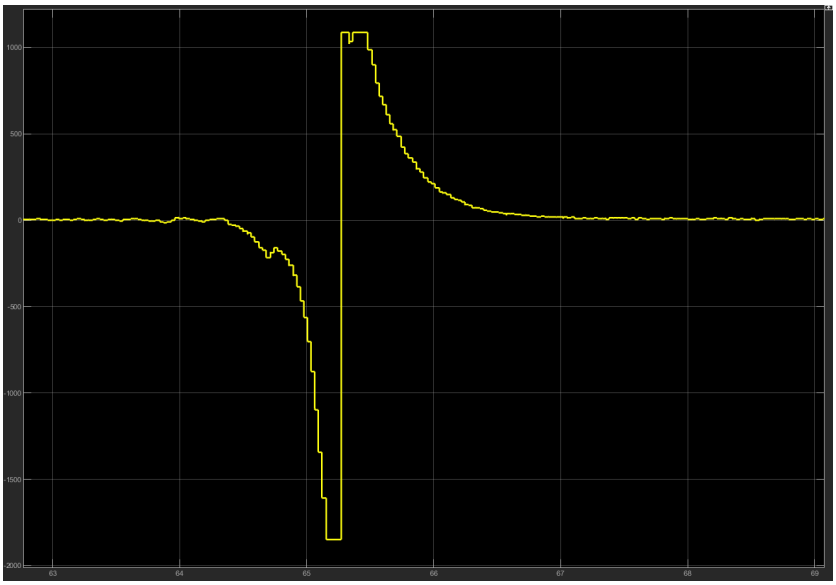


Figure 42 - gyroscope reading

As can be shown, the angular rate increases in a negative direction, as the robot falls backwards. As the robot corrects itself, the angular rate will become positive and decrease to 0, as the robot moves in the other direction, working to correct itself (see Figure 42 - gyroscope reading)

However, to receive angle, many methods were tested to get the most accurate reading.

Firstly, as explained above, the angle can just be taken as a reading from the accelerometer. This was a very easy method to obtain a value for angle. However, it had its drawbacks. The signal was shown to be very noisy.

To add to this problem, it was noticed and discovered in research that the horizontal acceleration of the robot would affect the reading of the angle. This was due to the motors driving the robot forward and the accelerometers inability to distinguish the difference in acceleration and gravity.

Introducing a low pass filter

An efficient and easy way to approach this problem was to pass the accelerometer reading through a low pass filter. Simulink's own low pass filter was used and the parameters were set as shown below (see Figure 43 - Lowpass filter).

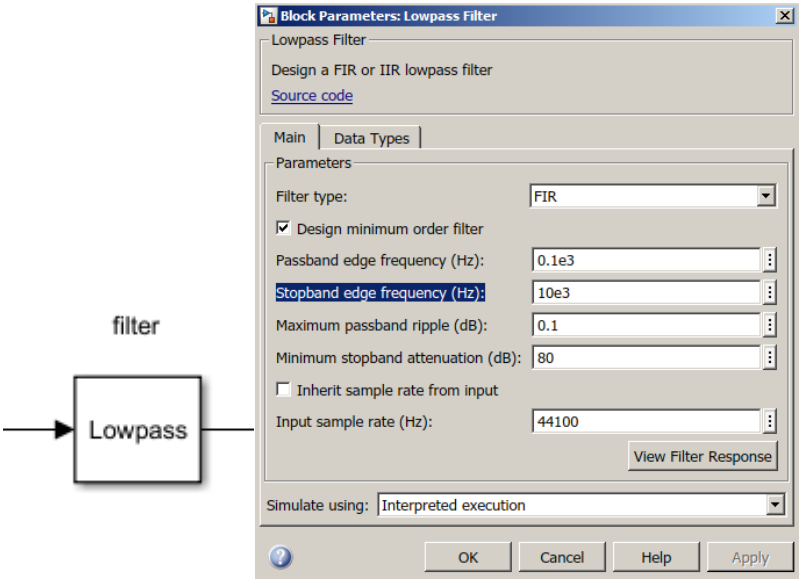


Figure 43 - Lowpass filter

A low pass filter takes the signal coming from the accelerometer and filters out the short-term fluctuations in horizontal acceleration allowing only the long-term changes, in this case gravity, to pass through the filter.

The drawback to using this method was the lag it introduced to the system. It was found that the more that the system filtered, the more the system would lag as shown below (see Figure 44 - Accelerometer signal (yellow), filtered signal (blue)).

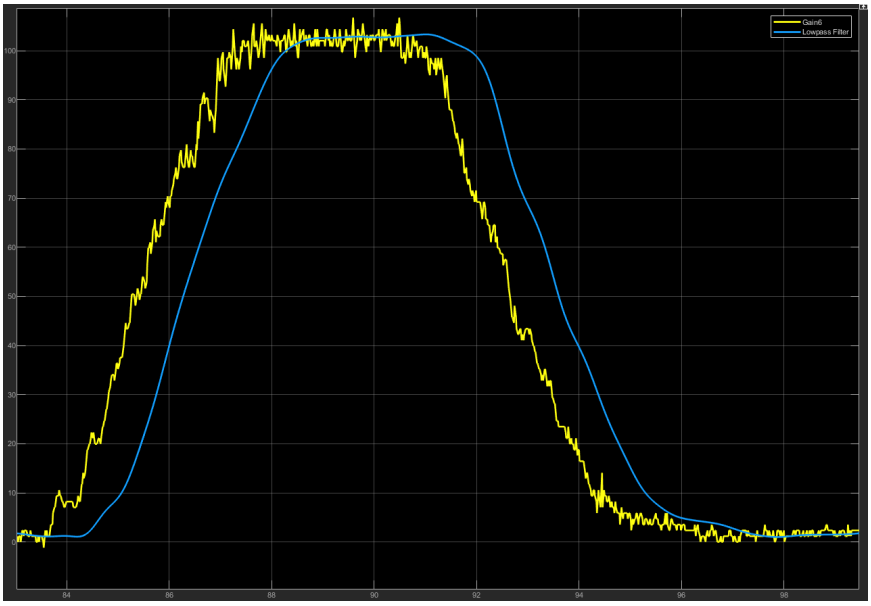


Figure 44 - Accelerometer signal (yellow), filtered signal (blue)

Single sensor testing

It was found possible to use only a gyroscope in the system to get a reading for angle and angular velocity.

The angle of the system can be worked out through numerical integration of the angular velocity. This also proved to eliminate the inaccuracy caused by horizontal accelerations. However, major gyroscopic drift occurred. This occurred because the reading from the gyroscope constantly fluctuated when the robot was still and the integrated fluctuations added to give a reading for angle that would become increasingly inaccurate (see Figure 45 - actual angle (yellow), example of gyroscopic drift (blue))

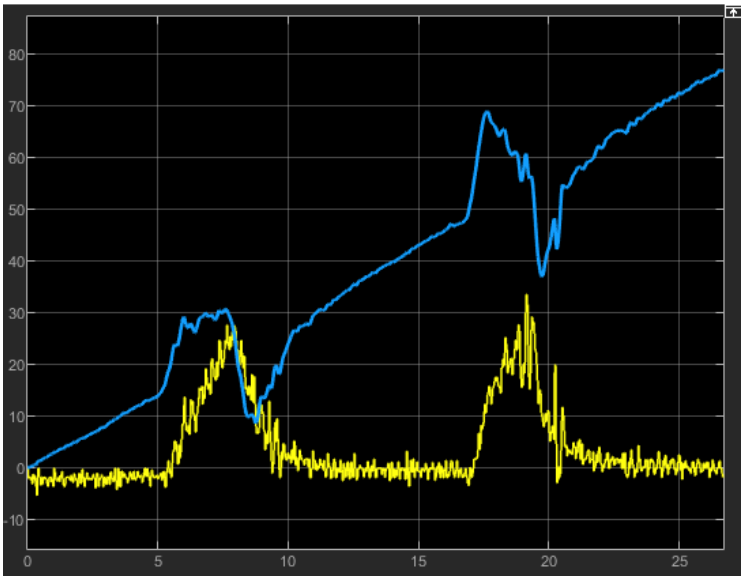


Figure 45 - actual angle (yellow), example of gyroscopic drift (blue)

Complementary Filtering

The complementary filter was set up by using a combination of the previously stated methods. A low pass filter was firstly used on the accelerometer.

The gyroscope signal was then integrated, to give a value for angle and this was put through a high pass filter in an attempt to eliminate gyroscopic drift.

These two signals were then summed in a summing junction to give a more accurate approximation for angle, although a high pass filter was also put in place, in theory, to eliminate gyroscopic drift. Gyroscopic drift still occurred and so the gyroscope was not used to obtain a value for angle.

The reading for angle was simply taken from the accelerometer and put through a low pass filter block on Simulink. This was due to gyroscopic drift occurring any time that the signal from the gyroscope was integrated.

4.2.2 Motor Encoder Readings

The results for position from the motor encoders proved to be effective but when initially testing for speed, accurate data was not given (see Figure 46 - inaccurate speed data from encoders).

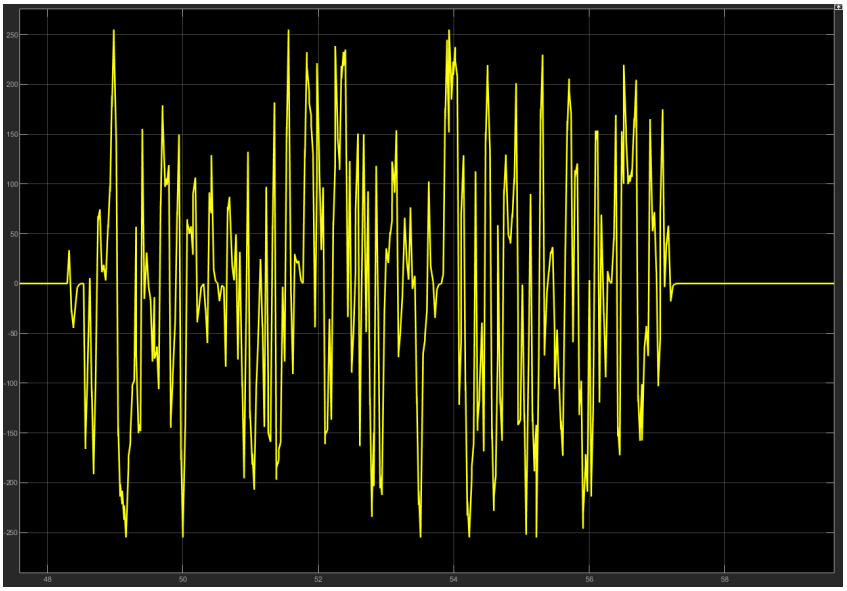


Figure 46 - inaccurate speed data from encoders

This was due to the fact that the sample time the encoders were set to read was too slow, because of this when the signal was differentiated the output gave a very noisy reading.

To counter this the sample time was sped up to a value of 0.000001 seconds. This crashed the system when running the model though Simulink, as the data could not be communicated from the Arduino board to the computer efficiently.

Although this was the case when the program was deployed to the Arduino board, the program was found to run smoothly and give good results.

The speed at which the motor encoders slow the motors down, could be adjusted by changing the value of the gain blocks attached to the system.

4.2.3 Stability of the System

Once the system was created in Simulink the stability of the robot was tested.

To understand the different components of the system, the robot was tested using individual components.

To begin with, the system was tested using just the accelerometer as a controller. When the robot starts to fall, the angle will send a reading to the motors telling them to drive to correct the position of the robot.
This was found to be unsuccessful, due to the signal given not being strong enough to drive the motor at lower angles.
The accelerometer signal was passed through a gain block to increase the amplitude of the signal going to the motor.
This allowed the motors drive at a higher speed at lower angles.

To help balance the robot, the gyroscope was used in conjunction with the accelerometer. This gave the system a kick pulse when the robot started to fall. (See Figure 33 - Combined gyroscope and accelerometer data).
This helped to bring the robots angle back to 0 faster and it was found that whilst the angle was positive, and the angular rate was negative. The motors would slow themselves down in advance (Shane Colton, 2007)

This, in theory, should have sped the robot up to catch its fall and smoothly bring the robots position back. However, it was found that the response was not fast enough to catch the robot, before it fell over.
It was found that gyroscope was very noisy even when stationary and that when the gain was too high, the system would become very shaky.
To accommodate for this, the gain value attached was decreased to eliminate this effect but not lowered so much that the motor would not respond fast enough to the kick.

The combined signal of the gyroscope and the accelerometer were then put through a PID block. The proportional value was increased, to get a better response from the motors. However, the rest was left alone, as the integral value was found to slow the system down too much and the derivative was found to make the system very noisy and shaky. In the end this was replaced by a regular gain block.

To attempt to balance the robot, the values in the system were tweaked and with each change the system was tested. Through intense testing the robot would not balance and proceeded to fall over with each attempt.

5. Discussion

The results from the physical model do not replicate that of the simulated model as it was not possible to balance the physical model.

The simulated model.

Although the simulated model balanced out, it was shown that the system still took 8 seconds to stabilise. This was not deemed fast enough. The system was tested further by re-evaluating the values for the gain blocks, however no better results could be obtained. It was not found how conclusive the Simulink model was due to the inaccurate magnitude of the y axis scale.

The scale that represented the angle (y axis) at which the robot tilted, was also deemed too small.

It could be argued that the simulation was not accurate or fast enough to respond for the following reasons

- The robot was modelled as an inverted pendulum in one direction
- Friction was not considered in any calculations
- Inaccurate CAD model
- Mathematical model and block diagram

The fact that the robot was modelled as an inverted pendulum was verified in many academic papers (see section 2. Literature Review). The robot only moves in one plane therefore it would not have made sense to model it in another plane.

Friction occurred in the motors and between the wheels and ground. This was completely ignored in all the calculations. It cannot be said how much impact this would have on the system. However it was assumed to be too small to make any impact on the system.

As the CAD model was not precisely modelled, it was thought this could have caused some errors. However, the values given by the software were approximately equal to those measured and so this idea was discarded.

It was deemed that the reason the model was incorrect was due to the block diagram in Simulink being incorrect.

The physical model.

The physical model was not able to balance and after huge amounts of time changing the program no balance was obtained. The system's gain block values were constantly tweaked and tested in order to try to obtain balance. As well as this, many attempts for more accurate readings were tried as shown in section 4.2.1 Sensor Readings.

It could be argued that the physical robot was not able to balance because of the following:

- Inadequate motors
- Sensor inaccuracy

The EMG30 motor set used in the project caused many problems, that could have accounted for the failure of the robot's ability to balance.

The first problem with the found with the motors was the power. If a value of less than 100 was put into the PWM block on Simulink the motor would not speed up. The motor was only found to run if the value was higher than 100.

This problem could be countered somewhat by adding gain blocks to the system however this gave them a small speed range.

The motors were also found to cause a lot of vibration and due to this caused a lot of interference with the angular sensors. This would create a lot of noise from the angular readings and cause the robot to rattle as this was read by the motors.

This rattling would also cause a lot of motors to break. A total of 3 replacement motors were needed for this project as the gears inside would break, creating too much play in the motors.

The angular sensors proved to give accurate data but as stated were hugely affected by the vibrations in the system. This could have caused a lot of inaccuracy.

It was also found difficult to determine if acceleration of the robot was completely eliminated from the accelerometer readings.

As the motor encoders could not display a reading for angular velocity it was difficult to determine whether the speed controller was working.

6. Improvements and Further Work

In order to improve the results given in both the simulation and the physical model the following changes would have to be made:

- Re-evaluate the mathematical model
- Replace the motors
- Replace the wheels
- Remove vibrations

The mathematical model would have to be re-evaluated, to acquire a more accurate scale for the angle output. Once this was achieved, a better alternative for a controller for the system would have to be found to achieve a better result for settling time.

A possible option for modelling the robot is the SISO (single input single output) tool. Using the SISO tool the user is able to create an architecture and from this import transfer functions into the values for G1, G2, H1 and H2.

From this, the SISO tool can be used to design a controller using Interactive Bode, root locus, and Nichols charts. As well as this it would be possible to use PD tuning to get values for the compensators used in the multi loop control architectures.

This information could in turn give a better understanding of how to stabilise the physical model.

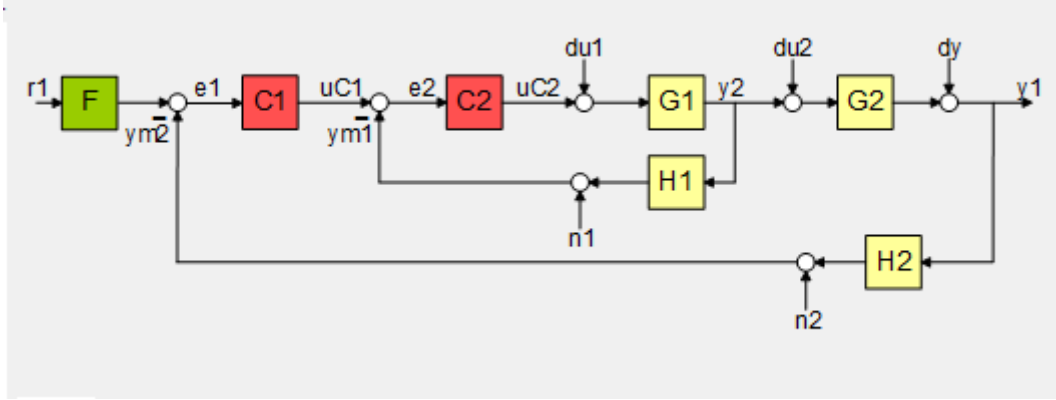


Figure 47 – Architecture option for the self balancing robot.

The motors did not provide a good output and so it would be worth investing in another model with a higher torque, so that at lower values of pulse width modulation the wheels will turn. More research into calculating the motors speed would have to be done, in order to perfect the speed control applied in Simulink.

To improve the stability of the robot, it could be suggested that wheels with a larger radius be used. This would decrease the time that it took for the robot to move positions which could improve stability.

In theory in order to fix the vibration, it could be suggested that dampening materials be used to mount the angular sensors to. More research into this would have to be done.

7. Conclusion

The self-balancing robot proved to be a difficult project and even though the system looked basic at first, it gave the opportunity to study and illustrate a lot of different techniques and concepts from sensing to modelling to control to actuation: the whole spectrum of mobile robotics. The project also added to mechanical and electronic engineering when building the physical model.

The final controller required more calculation and a deeper look at how fast the motors must spin to correct the deviation from the vertical as this could not be shown through Simulink.

The project proved that with more work - at recovering the deviation angle and the motor speed from the sensor values, and having the motors run at a specific speed - a basic PD controller would work. Using good sensor post-processing chains, where the values of the deviation angle and the motor speed are accurate could keep the deviation angle to zero.

A deeper look at the mathematical model would be useful when simulating the project.

References

CANETE, L. & TAKAHASHI, T., 2015. Modeling, analysis and compensation of disturbances during task execution of a wheeled inverted pendulum type assistant robot using a unified controller*. Advanced Robotics.* **29**(22), pp.1-10. Available from: 10.1080/01691864.2015.1070106.

FERDINANDO, H., KHOSWANTO, H. & PURWANTO, D., 2013. Performance Evaluation of MMA7260QT and ADXL345 on Self Balancing Robot*. Telkomnika.* **11**(1), pp.1-10. Available from: 10.12928/telkomnika.v11i1.258.

FRANKOVSKÝ, P., DOMINIK, L., GMITERKO, A., VIRGALA, I., KURYLO, P. & PERMINOVA, O., 2017. Modeling of Two-Wheeled Self-Balancing Robot Driven by DC Gearmotors*. International Journal of Applied Mechanics and Engineering.* **22**(3), pp.739-747. Available from: 10.1515/ijame-2017-0046.

HA, Y. & YUTA, S. *Trajectory tracking control for navigation of the inverse pendulum type self-contained mobile robot.* , 1996.Robotics and Autonomous Systems. Available from: //doi.org/10.1016/0921-8890(95)00062-3.

HANAFY, T.,O.S. & METWALLY, M.K., 2014. Simplifications of the Rule base for the stabilization of Inverted Pendulum System*. TELKOMNIKA Indonesian Journal of Electrical Engineering.* **12**(7), pp.5234. Available from: 10.11591/telkomnika.v12i7.5358.

HENRYRANU PRASETIO, B. & KURNIAWAN, W., 2018. Disturbance rejection using feed-forward control system on self balancing robot*. MATEC Web of Conferences.* **154**, pp.. Available from: 10.1051/matecconf/201815403002.

MODESTUS OLIVER ASALI, FERRY HADARY & BOMO WIBOWO SANJAYA, 2017. Modeling, Simulation, and Optimal Control for Two-Wheeled Self-Balancing Robot*. International Journal of Electrical and Computer Engineering.* **7**(4), pp.2008-2017. Available from: https://search.proquest.com/docview/1934328238?accountid=15977.

PARK, J. & CHO, B., 2018. Development of a self-balancing robot with a control moment gyroscope*. International Journal of Advanced Robotic Systems.* **15**(2) Available from: 10.1177/1729881418770865.

RAHMAN, M.D.M., RASHID, S.M.H., HASSAN, K.M.R. & HOSSAIN, M.M. *Comparison of different control theories on a two wheeled self balancing robot.* , 2018.AIP Conference Proceedings. American Institute of Physics IncAvailable from: 10.1063/1.5044373.

RIASAT, Z., IQBAL, T., ADEEL, U. & KHAN, M., 2013. Two Wheel Self Balancing Robot*. International Journal of Technology and Research.* **1**(4), pp.130.

Control Tutorials for Matlab and Simulink. 2017. Inverted Pendulum: System Modeling. [ONLINE] Available at: http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum§ion=SystemModeling

Shane Colton. The balance filter. http://goo.gl/WR8e6m, 2007.

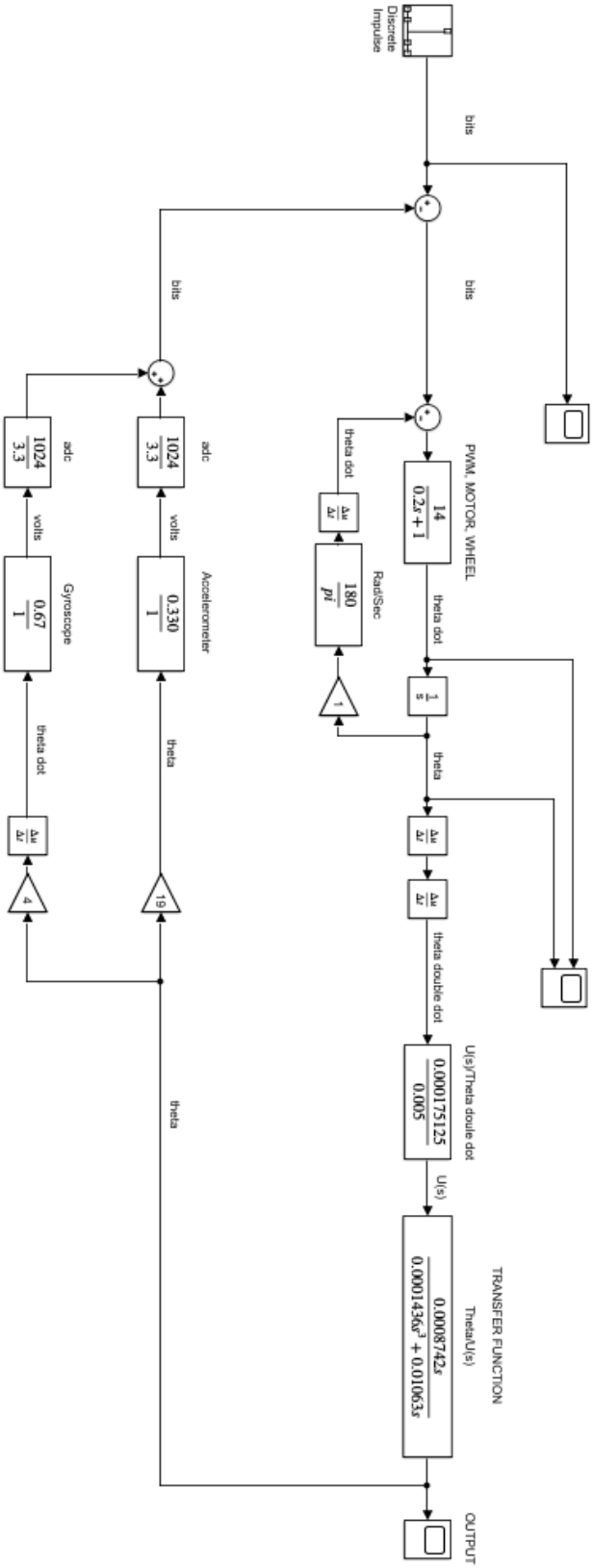
Memarbashi, H. (2010). Design and Parametric Control of co-axis Driven Two Wheeled Balancing Robot.

HELLMAN, H., 2015. Two-Wheeled Self-Balancing Robot. Bachelors. KTH Royal Institute of Technology: KTH Royal Institute of Technology.

Ogata, K., 2008. Modern Control Engineering: International Edition. 5th ed. [Unknown place of publication]: Pearson.

Appendices

Appendix 1.1 - Full closed loop Simulink Model



Appendix 1.2 - Full Simulink Program