Carleton UNIVERSITY | Faculty of Engineering and Design

SYSC 4805 - Computer Systems Design Lab

Final Report

Group #9

Students:

Theodore Hronowsky 101008637
Emmanuel Oluyomi 100953361
Zein Hajj-Ali 101020677
Hussain Aljabri 100935515
Ahmad Ayyoub 100954214

Table of Contents

Abstract

The SYSC 4805 Self-Balancing Robot project sets out to solve a predefined task through the use of a robot platform. The Self Balancing Robot independently balances itself without any auxiliary aid. The robot does so using components such as motors, motor drivers, an absolute orientation sensor and an Arduino Nano, all positioned on a custom printed chassis. A PID control system was developed, alongside self balancing algorithms to achieve this feat. The group intended to add a line following aspect to the robot, however due to complications and time constraints this was not achieved. The solution can be applied to two-wheeled personal transportation methods like the Segway and the Hoverboard. It can balance itself, and adjusts its correction algorithm to balance with extra weight put on it.

1.0 Project Details

Section 1.0, project details, includes brief summaries of information that is crucial to understanding the project. It discusses the problem background, our problem statement and our proposed solution.

1.1 Problem Background

This final report responds to the request on the achievements and technical aspects of the SYSC 4805 Computer Systems Design Lab project. This project aims to design and implement a robotic platform that with the aid of sensors and software, solves the task of self balancing. The robot has been designed as a real robot platform rather than using simulation software. This platform has been implemented using a robot kit as a base, with modifications applied in order to achieve a self balancing profile.

This is the schematic sketch of what the robot will do when it starts to move to balance itself:

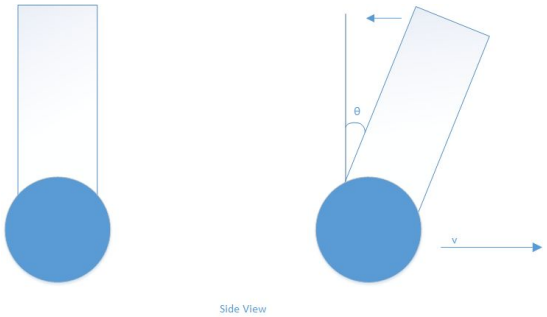


Figure 1: Self-balancing problem [1]

1.2 Problem Statement

The problem statement for the project is to develop a self-balancing robot that can overcome external forces by remaining upright as it moves. This problem is for a small scale setting, however it can be in a large scale setting which solves the problem of carrying a passenger and moving them from point A to point B over a short distance in a period of time. Many larger scale applications are derived from this, and are discussed later, in Section 5.3.

1.3 Proposed Solution

A self-balancing robot will be developed to move independently and will be able to balance itself while moving without falling over or have any external force driving it. Each time it leans forward, the robot moves forward and when it leans backwards, the robot moves backwards. The combined actions will result in a robot that can achieve self balancing. This will be done by applying self balancing algorithms as well as a PID system, that given input parameters, will modify its behaviour to correct its orientation.

1.4 Overview

The remainder of this report will cover the solution to developing a self-balancing robot, which will include the implementation process algorithms, PID control system, applications, project management and final outcome of the project. Section 2.0 will cover the project details, such as the achievements of the group, any challenges encountered, as well as the components that were used to complete the robot. Section 3.0 covers the technical sections of the project. This includes the implementation of the self balancing algorithms and overall solution, as well as the PID control system used. More so, this section will describe the theoretical aspects of the control systems and how the values are generated and used. The project management used, Section 4.0, will explain how the project was managed and the team contributions. Section 5.0 will include the final outcome of the project and will outline the applications of the project as well as any recommendations, moving forward. Lastly, an appendix is included for any relevant and important figures and supplementary material that is necessary to understand the project fully.

2.0 Project Discussion

Section 2.0, project discussion, focuses on the achievements of the group. While all projects are subject to setbacks and challenges, our project is no exception. This section also highlights the challenges that our group encountered and deviation from the original proposal. This section also discusses non technical aspects such as the components chosen for the project.

2.1 Achievements

As the SYSC 4805 project has come to an end, the self balancing robot has been developed. A 3D printed chassis was designed to house all the components and self balancing

algorithms were developed to keep the robot upright and self balancing. A more detailed breakdown of the group's success and achievements can be seen below.

Since starting the development of the self balancing robot, the following has been achieved (in chronological order):

- Research of components needed as well as various fundamentals needed to achieve self balancing
- Acquisition of all necessary components
- Design and 3D printing of structure that will be used as the body of the robot, which will hold and connect all the components together
- Framework of code has been put into place as well as the general plan of development
- Code management has been implemented to facilitate the development process (GitHub)
- Testing of various components to ensure they work as intended
- Overall assembly of the robot
- Beginning stage of developing the algorithm to balance the robot
- Self balancing algorithms completed.
- Testing phase to determine precise parameters for optimal performance.
- Final demonstration of the self balancing robot.

The assembled self balancing robot is depicted in figure 2, shown below.

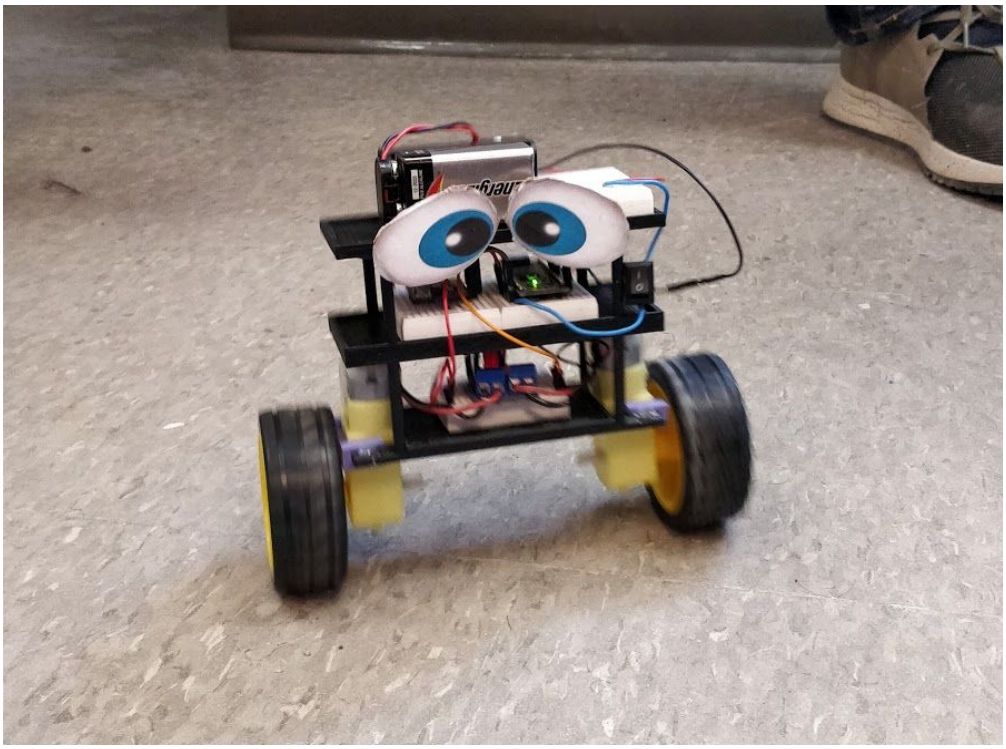


Figure 2: Self balancing robot assembled

2.2 Project Setbacks and Challenges

In any project, it is expected that there will be setbacks or challenges encountered. The setbacks that the group has come across have mostly developed from the hardware used. Many sensors that were provided to us in the robot kit had proved to be faulty or non functional. For example, it was determined that one of our motors only drove in one direction. This is due to a faulty motor driver. In a design where the robot needs to be able to move back and forth to balance itself, a fully functional motor is necessary. In addition, it was determined that the type of motors used provided a greater challenge. Because the speeds of the motors were not precise, it resulted to the difficulty in keeping the robot balanced. Also, accessories such as jumper wires were not provided in a sufficient quantity in order to complete our design. These setbacks resulted in the group having to spend time sourcing new parts. In addition to this, many unexpected complications arose during the project. For example, during a testing session, the provided AAA batteries died midway through the testing. This ultimately brought the session to a halt, until new batteries could be acquired.

More so, related to our battery issues, the group faced challenges powering the robot. It was determined that the supplied battery pack was not sufficient to fully power the robot. As a result, we started to use a micro usb in addition to the battery pack. Because the micro usb that we had was short, it limited the mobility of the robot. A longer micro USB cable was then used, however, this longer cord had a weight effect on the robot and it was determined that a cord could not be used. The group then decided that on board batteries was the best option and dealt with weight distribution problems until an equilibrium of the systems weight was determined.

In terms of software, one challenge that arose was with the Adafruit BNO055's required library not supporting features that we needed. In our case, the library did not support hardware interrupt output pins that were needed for the sensor. In addition, we faced the issue of our absolute orientation sensor data deviating and drifting over time.

When it came to the final demo, the team experienced challenges with the battery system. It was planned and anticipated that the group would keep one fresh 9v battery aside while testing, so that it could be used for the final demo. When it came time for the demonstration, the new battery was attached, however, the robot had troubles powering itself. While this issue wasn't the first time it presented itself throughout the project, it was unfortunate that it did in such an important time even after it had been anticipated.

2.3 Deviation from project proposal

The plan that the team followed through until the end of the project remained similar to the groups initial plan, outlined in the project proposal. However, there were some minor issues that led to change, improvement of or use of different products, sensors and other components needed to build the robot.

For example, in the project proposal the group was to use seperate chips for both the gyroscope and accelerometer. Instead, the group decided to use the Adafruit BNO055 absolute orientation sensor. A more detailed description of this component can be found in section 2.4. By combining 2 chips into one, we cut back on the amount hardware needed to be fit on the 3D printed chassis, which ultimately made the design lighter.

We also had issues with the initial kit. Because of that we decided to change the whole kit and sensors to fit our project needs. This ensured that we had a functional kit with all the components in it, and in a sufficient quantity for our needs. We changed the kit with the help of Graham Eatherley.

The group also intended to implement a line-following aspect to the robot in the original plan. This was only to be added if there was sufficient time after completing the self balancing. This feat would be accomplished using IR sensors, positioned on the front of the robot facing downwards to the floor. A bracket part was developed to hold the IR sensors to the front of the robot allowing the orientation of the sensors to be modifiable. Line-following algorithms were also developed however, due to time constraints, were not included in the final product. This is mainly because of the way the robot balances currently. Adding the line following algorithms would prove to be too time costly and extremely difficult in our situation.

2.4 Components

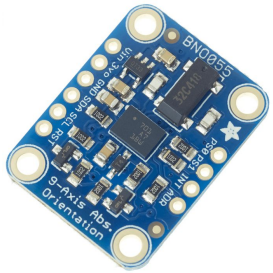
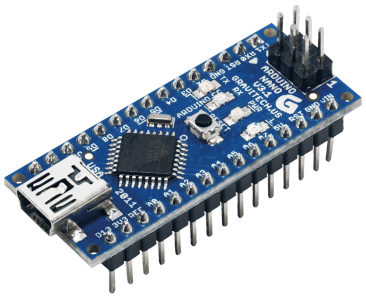
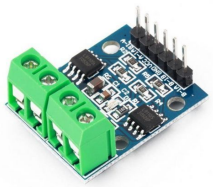
The self balancing robot is made up of many various components and sensors. The main component of the robot is the 3D printed structure. As the existing platform was not designed in a vertical form, a new vertically oriented structure was 3D printed. This 3D printed structure is depicted below, in Figure 3. A picture of the CAD drawing is depicted in Figure 15 of Appendix F. The structure is designed to have three level layers which can carry the other components for the robot like all the breadboards, batteries, motors, absolute orientation sensor, wirings and the connections. The structure has a small lip on each layer in order to secure the components, ensuring that they do not fall out. More so, the components are 2 way taped to the structure so that they do not come loose, which could lead to detached connections. The topmost layer will carry the batteries as that is the heaviest component we have, this will cause the robot to balance

itself while carrying the batteries. There will also be a miniature breadboard on the top layer to connect the switch to the batteries and rest of components. On the middle layer, the Arduino and absolute orientation sensor are housed. On the bottom layer the motor drivers, along with the motors and wheels are attached. The design was modeled and designed similar to that of the CARL5 so that the motors would fit on without any altering. Because the structure is 3D printed and is composed of plastic filament, this structure is very light in weight so the robot only focuses on balancing the weight carried.

Figure 3: Figure of the 3D printed structure

In addition to the 3D printed structure, there are 3 main sensors used in the design. They are depicted in Table 1, below.

Table 1: Components

(a) Adafruit BNO055 [2]	(b) Arduino Nano [3]	(c) Motor Driver [4]
		

Component A (Adafruit BNO055): The Adafruit BNO055 is a absolute orientation sensor meaning it has a gyroscope and accelerometer on board. This sensor allows for 4 point quaternion output, which means that our data manipulation will be more accurate. The gyroscope senses the angular velocity of the robot. This allows us to sense the rotational motion and changes in orientation allowing us to alter parameters to ensure the robot stays upright. The accelerometer measures the non-gravitational acceleration of the robot. This is useful for the implementation of line following. This sensor is used to determine the acceleration of the robot in a particular direction.

Component B (Arduino Nano): The component that controls and connects all these hardware devices together is the Arduino Nano. The Arduino Nano is a microcontroller designed by Arduino. It contains an Atmega328 controller which contains the program necessary to carry out the functionality of the self balancing robot. This controller was used because of its small size, minimal power consumption and flexibility. This is integral for any embedded systems design and is excellent for robotic applications due to its specialized library functions for robotics.

Component C (Motor Driver): The motor driver is what drives each individual wheel of the robot. Through modifications, produced by real time results, the robot will use forward or reverse movements to counteract and self balance itself. It is through the motor drivers in which this is achieved.

In addition to these 3 mentioned main sensors, there are intermediate components. For example, there is the battery pack which holds 4 AAA batteries to supply power to the robot. This battery is connected to a switch. We added another 9V barry late in the design phase to make up for our power needs. There are also 2 motors that are connected to wheels which are used to roll and move the robot. All these components are connected using jumper cables. More so, the use of IR

sensors was to be integrated into the design. This was not added, however the group spent considerable time on this subject. By using IR sensors, we would have been able to guide our robot along a line, achieving a line following aspect while simultaneously self balancing. The IR sensors were planned to be mounted on a custom bracket that was designed so that the IR sensors can be positioned in various positions, allowing for line following capabilities of lines of any width.

3.0 Technical Aspects

In this section, section 3.0, the technical aspects of the self balancing robot are featured. It explains the use of the PID system, an integral part of our solution. It also discusses how our data is acquired and processed.

3.1 Implementation and Solution

In the early phases of the project, many project management techniques were used. Figure 12, in Appendix D, shows the work breakdown structure for the project. Prior to development, or the implementation phase, project management and design activities were conducted. Once these preliminary steps were completed, and the design of the robot was determined, it was time for the implementation phase.

First, a vertical profiled chassis was custom 3D printed, along with the acquisition of all necessary components. This was the best course of action because the platform given to us in the project kit could not be repurposed without destroying it irreversibly. More so, a deposit was put down for the kit and the group wanted that back. The robot was then assembled on the chassis and the components were mounted and wired up. A detailed schematic of this can be viewed in Figure 13 of Appendix D. This led to the software side of the solution. Framework for the self balancing robot was researched. A lot of existing code was available online, however the group wanted to implement our own algorithms, from the ground up. Planning and design of the algorithms was created and many diagrams were created to aid the groups through process. Figure 4, below, depicts a flowchart that describes the general flow of the self balancing algorithm. This diagram has also been included in Appendix D for quick referencing.

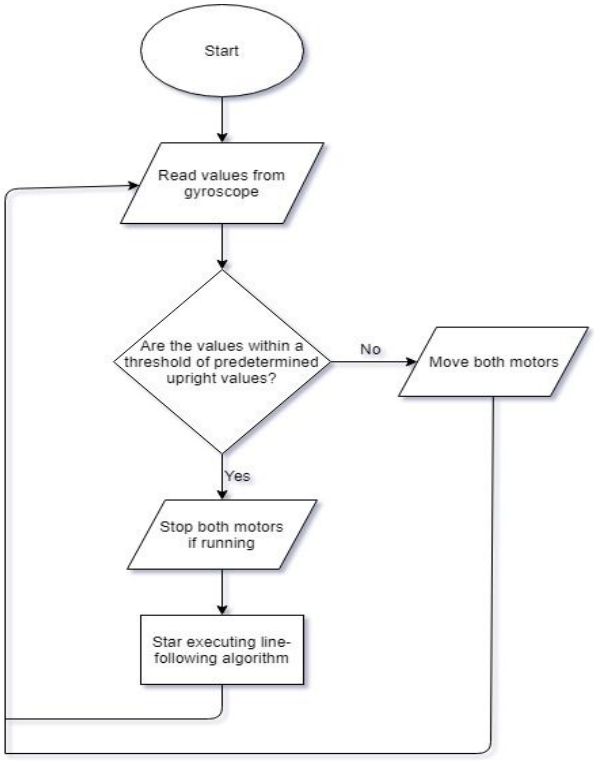


Figure 4: Flow chart for self balancing algorithm

To explain the above diagram, we will be predefining values for the arduino to compare against and tell if it is standing upright. We get these values basically using trial and error, but once they're set, they should work consistently. It will read from the absolute orientation sensor the quaternion values, and compare them to the predefined values. It will then decide whether to move the motors forwards or backwards and re-poll the absolute orientation sensor again. Once it determines that it is upright, it will turn the motors off. And the process starts again. Since it will never be 100 percent upright and balanced, it will keep polling and adjusting as needed.

Based on the above stated theory and reasoning, the development of the framework was coded, which then led to the testing phase. Various parameters that dictated the output of the robot were manipulated through trial and error. This process is outlined in Section 3.4, testing. Threshold values to keep the robot upright without excessive wobbling were measured, calculated and reflected in the self balancing code. This was once again obtained using trial and error as this is the easiest and only method to acquire these values. The values off of the COM port serial monitor of the Arduino IDE helped us to refine and obtain these values. Through excessive iterative testing and modification of the PID values, the self balancing algorithm was refined and a smoother output was obtained. Once the desired performance was achieved, the feat of self balancing was said to be completed.

While the final product does not include the line following aspect that the group had originally intended to complete, the implementation of this portion is included, due to the extensive time that was spent on it.

The flowchart in Appendix D, depicted by Figure 10 describes the basic line-following algorithm that was to be executed simultaneously with the self-balancing one. It simply reads from the IR sensors, and drives in the direction needed to keep the sensors straddling the line. We did not have the time to test the line-following algorithm running with the self-balancing one, and therefore, the algorithm may be subject to more modification. Various code snippets of this are included in Appendix D.

At its final stage, the self balancing robot achieves its set out feat to self balance. A more detailed discussion of the outcome of the project is discussed in Section 5.1.

3.2 PID System

In order for the self balancing feature to work, a PID system was implemented into the design. P.I.D stands for Proportional, Integral, and Derivative. A PID system is a feedback loop controller which helps to produce results similar to a set point, irrespective to any variation or disturbance. Output is calculated based on measured error and the gain of the system, which will be discussed further on. Using this system we can create a correction response to drive the motor to keep the robot balanced and upright. We initially feed the current angle to the system, and it will be compared to the targeted angle resulting in the creation of a response signal to drive the motor. The proportional part will take care of the present, and produce a response that is suitable for the present position. The integral portion produces a response based on the past readings and will carry out past situation to present. Derivative gives a prediction of the future, and takes the future situation into consideration. All three functions together will produce a response that will make the robot go back to the balancing position without tipping over or falling. The following figure, Figure 5, shows a block diagram of how the PID system handles an input and how it achieves its result.

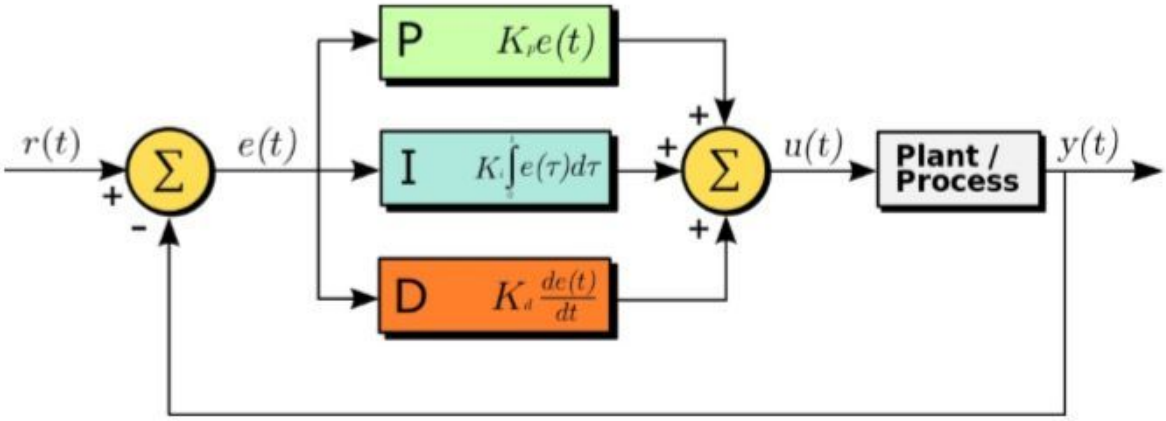


Figure 5: Block diagram for PID system [6]

The constants of the system can control the rise time, overshoot, settling time and steady state error. These constants are the proportional gain (Kp), integral gain (Ki) and the derivative gain (Kd). A good control system has a low rise time, settling time, peak overshoot and steady state error. This can be achieved by adjusting the constants. We only can adjust them by trial and error observations, adjusting frequently and gradually as we test. The table below, Table 2, shows how increasing each PID parameter changes the output of our system. During testing, we observed that if Kp increased (more correction) the robot will go back and forth much quicker. We know that a low Kp will cause the robot to fall over because there's not enough correction. For Kd, we know it is good value when the robot oscillates less than previous Kd value. And a good Ki value will shorten the time it takes for the robot to be balanced. Figure 6, below, shows the parameter values and step response for the self balancing robot.

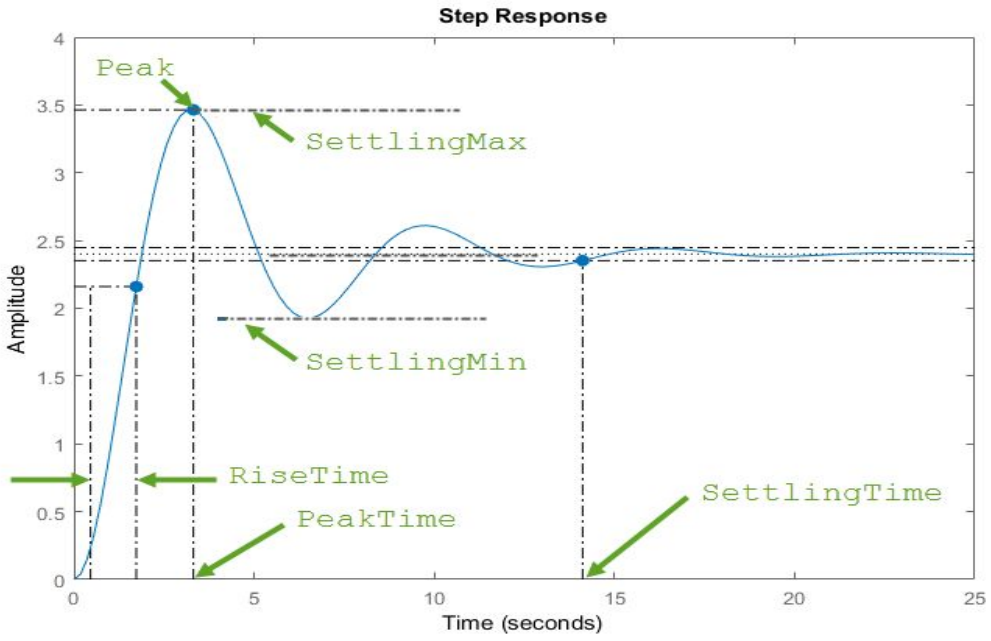


Figure 6: Parameter values for the self balancing algorithm [5]

Table 2: How PID parameters affect the system

Parameter increase	Rise Time	Overshoot	Settling Time	Steady-State Error
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Reduce/Eliminate
Kd	Small Change	Decrease	Increase	Small Change

As mentioned earlier, to help to determine many of our PID values, the COM port serial monitor and graph was used. Below, in Figure 7, an example of the COM port serial graph is given. In this specific example, we are monitoring the oscillations of the robot, which mimics the performance of the robots ability to self balance. These observations were then applied to the manipulation to our PID values either through increments or decrements to our last recorded value. Other parameters such as motor speed values, quaternion data (x,y,z,w), yaw, pitch and roll, and the output of the robot quantified as numbers were observed through these built in monitoring tools.

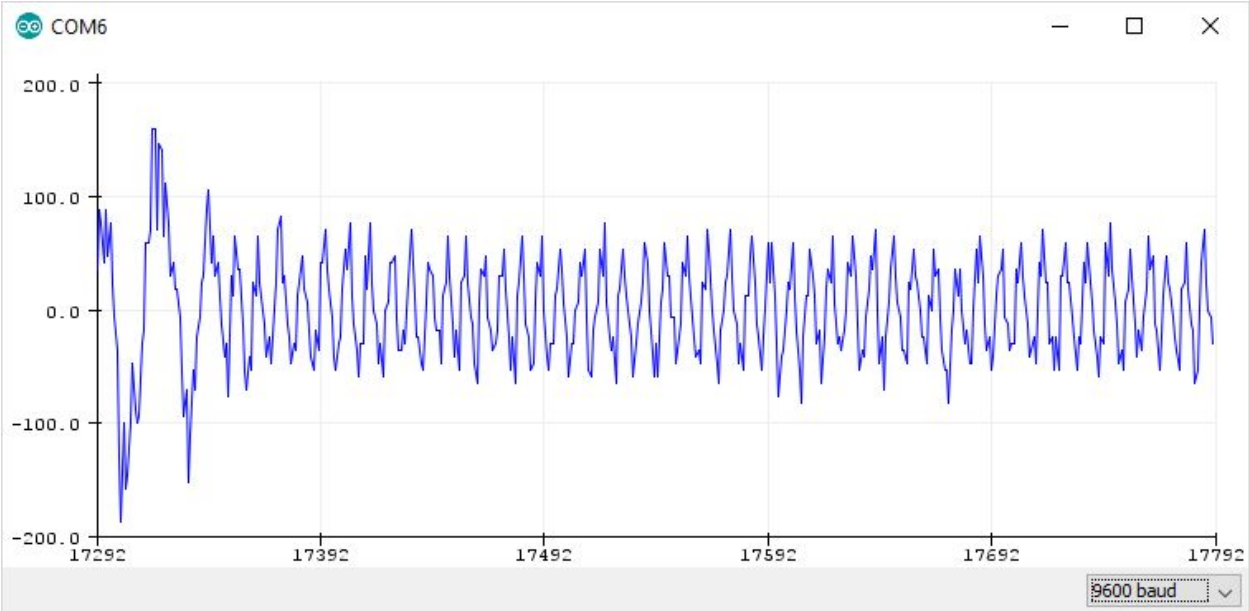


Figure 7 : Serial COM port graph

3.3 Data processing

The main data that we are focusing on is the data produced from the sensor that is responsible for balancing the robot, the absolute orientation sensor. This sensor's main responsibility is to provide us with the current coordinates of the robot such as x-axis, y-axis and z-axis values. This data will be constantly collected at every moment the robot moves to process it and to determine how can we improve the robot position in case of it encountering obstacles or things that might cause the robot to fall over. This data is also important to determine how fast the robot can go. If the robot is upright and responding well to the self balancing algorithm, the speed of the robot can be increased. If the robot is shaking or not properly balancing itself, then we will have to slow down the robot to avoid a crash or damage after a fall.

Another group of data that is important to be collected is the data from the line-following algorithm. As mentioned, while line following was not part of the final solution, the team spent considerable time researching/ developing and testing this, and figured that it should be included. This data tells us if the robot is properly following the line or not. If there is a deviation from the actual direction we will know if it is an algorithm problem or the IR sensor problem. In addition, the line following data is needed to know how fast we can drive the robot. Later on, the data will help us to make the robot move properly at the edges of the line. For example, the robot must be able to move right or left when it sees the end of the line or an edge on either side.

Lastly, the data that is needed to be processed is the data coming from the line-following and self-balancing algorithms. These data must be collected or delivered at the same time to be synchronized with the actual robot position or speed to be able to determine the best feedback for the robot. In other words, the data of line-following algorithm must not have a latency or delay with the corresponding data of self-balancing algorithm.

3.4 Testing

Once the self balancing robot was designed and assembled, it was subject to testing. Through various testing techniques, the output of the self balancing robot was optimized for our needs.

The first parameter that needed to be tested was the weight distribution of the robot. Because the design and placement of the battery was changed many times in accordance to the many battery variations we tried, the output of the robot varied. As additional batteries were added, it was observed that either adding the batteries directly over the center of the robots axel (center of gravity) or distributing the load of the batteries evenly was the best course of action.

Next, we began to optimize our PID control system, in charge or driving the self balancing algorithm. Through trial and error, Kd (derivative grain), Kp (proportional gain), and Ki (integral gain) was optimized to produce the desired output for the robot. While this is a trial and error process, there was a procedure that the team followed. This is listed below:

1. Firstly, the Kp, Kd and Ki values are set to zero and are the new starting and reference values for the system.
2. Next, Kp is adjusted. This was done by gradually increasing the value, and observing the output. This was done until the response to the disturbance was a steady oscillation. In our case the disturbance is the robot trying to fall over, or getting pushed by an external force, to an extent. The following was observed:
    a. Too small of a Kp value makes the robot fall over due to the fact that there is not enough correction.
    b. Too high of a Kp value will make the robot rock back and forth fast and unpredictably. This produced extremely large oscillations.
    c. The optimal Kp value will make the robot fluctuate back and forth slightly.
3. Once the Kp is set, the Kd is then adjusted. This will put the system in a critically damped condition and will reduce the oscillations. The following was observed:
    a. An optimal Kd reduces the oscillations to the point that the robot is in a steady state.

    b. More so, the optimal Kd value will allow the robot to keep balanced, even when subject to an external force attempting to push over the robot.
    c. Too high of a Kd will result in chattering of the system, meaning it will be subject to vibration.
4. Repeat steps 2 and 3 until the Kd fails to stop the system's oscillations. These values are now our Kd and Kp values that will be used to advance and configure the Ki.
5. Ki is then increased until the desired amount of oscillations is had. In many cases a value of 0 is used for a quicker response, however for our situation we had to increase this value. The following was observed:
    a. The optimal value will reduce the time it takes the robot to stabilize itself.
    b. The optimal value may produce so oscillation, but will reduce itself to a steady state.

After combining this process with the right weight distribution, a desired output for the robot was accomplished. Table 3, shown below, summarizes some of the findings of the group.

Table 3: Results of PID value refinement

Case	Kp	Kd	Ki	Output
Modification of Kp				
Case #1: Small Kp	10	0	0	Robot falls over easily and quickly. Does not even attempt to balance itself. Too much under correction
Case #2: High Kp	100	0	0	Movement is erratic and unpredictable. Moves back and forth wildly. Too much over correction
Case #3: Increase Kp from Case #1	25	0	0	Robot tries to balance itself to an extent. Less under correction.
Case #4: Decrease Kp from Case #2	75	0	0	Movement is less wild and a bit more predictable. Kp is determined to be somewhere in the middle of Case #3 and 4
Case #5: Kp is stable.	58	0	0	Kp value was found to be stable at 58. We will use this as our reference value moving forward.
Modification of Kd				

Case #6: Kd is set high	58	10	0	System is observed to vibrate/chatter. Less oscillations are observed.
Case #7: Kd is decreased	58	5	0	System is subject to less vibration. Again, less oscillations.
Case #8: Kd is decreased	58	1	0	Robot can be pushed over. Not enough oscillations.
Case #9: Kd is increased and is stable	58	2.5	0	Robot stays upright when pushed. Optimal amount of oscillation.
Modification of Ki				
Case #10: Ki is set low	58	2.5	10	Quick response, however not enough oscillation.
Case #11: Ki is set high	58	2.5	100	Not quick enough response, too much oscillation.
Case #12: Ki is found to be stable in middle of range	58	2.5	60	Optimal amount of system response and oscillation.
Case #13: Ideal (the one used)	58	2	60	Stable and consistent results, lead to self balancing for long duration.

As seen in Table 3 above, much testing and trial and error was needed to get the desired output of the robot. Another factor that contributed to testing was the speed of the motors. This was directly associated to the amount of power from our battery system. Due to the constant exponential decrease of our power supply over time, the values of Kd, Kp and Ki varied. For example, on a new, fresh 9v battery, the speed of the motors was high. After 30 mins or so, the speed of the motors was noticeably slower. It was found that on average, a 9v battery would last the group one hour until it was completely dead. This meant that the most of our productive testing had to happen within a 30 minute time frame of a new battery.

Some other parameters that contributed to testing and the refinement of our PID values were the threshold and setpoint of the robot. These values were crucial to determine the orientation of the robot. This was then used by the absolute orientation sensor to produce quaternion data which was processed by the PID control system. The setpoint of the robot is the angle in which the robot is desired to balance at and is an important value for the absolute orientation sensor. This value was tuned after much manipulation and was found to be an angle of 180 degrees. It is this value to which the robot is upright. Another value that played major

importance to our algorithm was the threshold. This would be used to determine within what range would the robot begin to notice that it was not upright. This value was set to 0, which meant that when it wasn't 0, or standing exactly upright, it would begin to employ the self balancing algorithm.

In addition to the testing of the self balancing algorithm, many additional programs were written to test the components. For example, programs to test the absolute orientation sensor, IR sensor, as well as the motor driver and motors were developed and used to test the output of the individual components. This proved useful to determining if the components were faulty, or if they were working to specification. Source code that shows the programs used to test are uploaded and can be found in the same zip file of this report.

A large portion of time was spent testing the output of the robot, between modification of the PID values. A great amount of videos were taken, which were then posted to the groups Slack conversation. The PID values were listed for each video as well and they were used for further review of the robot as well as for comparison. Because there were so many videos taken, only a select few of the videos, showcasing the extreme values and best possible outcomes are included in the zip file.

4.0 Project Management

Section 4.0 discusses the project management processes and techniques set forth by the team. Ensuring proper project management was put in place was crucial to the progress and completion of the project. Section 4.0 explains the individual contributions by each team member, various project control activities used, milestones of the project and the development tools used to facilitate the project process.

4.1 Contributions

For the most part, our team is split into two parts; support and technical**.** Part of the team, support, worked on creating and designing the 3D structure of the robot. Also, the support side worked on measuring the components or sensors sizes to fit the overall printed structure of the robot as well as setting the proper requirements for creating an exact size of the robot 3D structure. They have also worked on delivering the necessary and required project documents such as the proposal, progress report and final report. They have also worked on testing and assembly the robots components from sensors, self-balancing and line-following algorithms and integrations of the sensors.

The other team, technical, worked on defining the sensors that will be used in the robot. They have also developed the self-balancing and line-following algorithms. In addition, they have developed and tested the combination of the both algorithms. Furthermore, they have faced challenges in developing libraries for the gyroscope sensor that is responsible for balancing the robot. In addition, they have encountered many challenges dealing with the hardware components of the robot from wires, batteries, motors and many other sensors.

It is difficult to list each members individual contributions, as the team worked as a whole in many areas. For example, when one aspect of the project required attention, all members directed their efforts to complete the task at hand. However, for the requirements of the SYSC 4805 project, a more specific and detailed description of each member's contribution based on work components is listed below:

- Theodore Hronowsky: Designed and modeled the 3D printed part and assembled the structure & hardware components. Worked on project deliverables and documentations. Final testing.

- Emmanuel Oluyomi: Designed and modeled the 3D printed part and assembled the structure & hardware components. Worked on project deliverables and documentations.

- Ahmad Ayyoub: Developed the line following algorithm. Worked on project deliverables and documentations.

- Zein Hajj-Ali: Developed the self-balancing framework and algorithm. Solved miscellaneous hardware and software problems. Final testing.

- Hussain Aljabri: Developed the line following algorithm as well as developed the self-balancing framework and algorithm. Solved miscellaneous hardware and software problems. Final testing.

4.2 Project control activities

In order for a successful project outcome, project control activities are important to implement and follow. From the very beginning of the project through to the end, many control activities were used.

For example, project planning steps were followed. This consisted of 7 steps to facilitate the planning and implementation of the sysc 4805 self balancing robot. A project planning diagram is depicted in Appendix A, Figure 8. This diagram outlines the project scope, schedule, resources, budget and cost, quality, risk and communication. The scope consists of determining the integral part of the project. The project schedule is the schedule that will be followed by the group. Project Resources consists of the components we will use and the acquisition of them, while the budget and cost deals with the cost associated with each component. Project risk entails understanding and planning for setbacks. Project quality contains testing and checking the robot with predetermined quality factors and requirements. Lastly, project communication deals with the coordination and task distribution of the group.

The team's process throughout the project can be organized into main milestones. The main purpose of the milestones are to organize our project into small work products. These milestones are divided into coding, designing, integration and reporting work products. These milestones are pictured in figure 9 of Appendix B, Control Activities. It describes the parts and overall integration needed to complete the project. From the picture and project control activities, the first and second circles represent the implementation of the line-following and self-balancing algorithms. The third and fourth circles represent creating, 3D printing and assembling the robot structure with the hardware components such as the sensors, motors and others.

The coding work product includes the line-following and self-balancing algorithms needed to drive the robot. In addition, there is a testing part where both algorithms are combined to balance the robot while following the line. Lastly, there is a part to check the quality and accuracy of the self-balancing algorithm when the robot is confronting obstacles or is put onto different circumstances and conditions.

The designing work product includes designing the main structure of the robot to hold all the necessary hardware components and sensors such as battery, gyroscope, motor and other various sensors.

Integration work product focuses on assembling the components and sensors, assembling the full picture of the robot with the 3D printed design, uploading the necessary coding algorithms to the sensors using the arduino chip, testing the robot after assembling it and uploading the code to it and finally validating if the robot is working as expected in the algorithm.

Lastly the reporting work product is mainly about organizing the necessary required reports from our team such as team progress report, team final report and others. This work product has scheduled deadlines for the team to group and write the required reports.

In addition to milestones to keep the team on track, weekly meetings were scheduled. This ensured that progress was being made regularly and every week. This also allowed for the all of team members to stay current and in the loop regarding the project. This was done by either meeting in the lab to progress on the software and hardware aspect, work on deliverables and meet to discuss as a team. The weekly meeting minutes can be tabulated, as seen in Table 4 of Appendix C.

4.3 Deliverables

In addition to dividing the project into milestones, the project was guided using deliverables. By submitting these deliverables in time, it allowed for the work of the project to be evenly distributed throughout the entire semester. The deliverables used to keep the team on track are as follows:

- Project proposal (Feb 1st): main details of project and robot platform to be used.
- Progress Report (Mar 3rd): current progress of report work
- Project Presentation (Mar 29th): Cover aspects of project (project management, solution, design etc).
- Project demonstration (Apr 5th): Demonstration of project.
- Peer review of individual contributions (Mar 29th) : Outline of contributions.
- Final Report (Apr 9th): Technical aspects of project and updated progress report.

4.4 Development tools

Many management and development tools and techniques were used to facilitate the project process. By implementing these techniques, the team was able to carry out tasks in an organized and thought out manner. Communication within a group project is key to success.

Using slack, a team based collaboration service, intercommunication between the group was performed. Using this application allowed for meetings between the group members possible. Integration of other applications within slack such as github and google drive, facilitated the team management. General discussion of the project was carried out through slack as well.

Github, a software development platform, was used by the group to manage the code produced. Github allowed for version control using Git as well as source code management. This

enabled the most current version of the team's code to be available to all members of the group. In addition, the use of branching off from the master allowed for individual tasks to be completed.

In terms of completing deliverables, Google Drive, with the use of Google Docs and Google Slides was used, enabling a collaborative environment to work in. The use of real-time, online editing meant that all members did not need to be in the same room at all times to work on the project deliverables.

Arduino IDE was used as the code development environment for the project. The environment allowed for writing and uploading of programs to the Arduino microcontroller. This platform was also used to write programs to test the functionality of the components.

Lastly, AutoCAD was used to design and create the 3D printed part. Using the program allowed for a structure that would work for our needs to be built. This design was then converted to a .stl file and sent off to the campus 3D printers for printing.

5.0 Conclusions and Considerations

5.1 Conclusion

In conclusion, a self balancing robot was designed and developed. A vertical profile chassis was developed and self balancing algorithms were developed from scratch. The final product satisfies the requirements of self balancing without any auxiliary aid. While it is not perfect, it is proof of concept and given more time and better components, could work more efficient. The final product, the self balancing robot, was showcased as part of a final demonstration and was able to self balance for a considerable amount of time, satisfying its set out, predefined task. The team faced many challenges from both software and hardware sides of the project but managed to overcome them through determination and teamwork. Source code can be viewed and accessed from: https://github.com/ZeinHajjAli/4805-selfBalancingRobot. In addition, videos and all relevant code will be attached in a zip file with this final report.

5.2 Lessons Learned

We have learned an extensive amount about project management tools and planning processes. Learning how to schedule our time to fit all the team members for meetings and how to efficiently use our time was a huge part of the project. In addition, we have learned how to create back up plans in case of facing any failures during our project development. Moreover, we

have learned to split the work as a team and update each other in case there are issues or solutions. Most importantly, we have learned that we should test our project components earlier to continue developing more components in our project. For example, we have encountered challenges creating the algorithm for self-balancing the robot. We tested it a bit late which causes us more delays. It delayed our next project component which is the line following algorithm. Our intention was to test the line-following algorithm with the self-balancing algorithm. However, it was more complicated than we expected. Therefore, we could not combine both. As a result, we have the self-balancing algorithm only implemented in the project. However, we have attached code snippets of the line-following algorithm for later references. These code excerpts are pictured in Appendix E, Figure 14.

In addition to this, we have learned about the uncertainty that comes with working with hardware. Because the group experienced many hardware related issues, it may have been a good idea to use a software such as V-REP to simulate the robot. This would make finding the PID parameters much easier. As well as would have eliminated any problems that come with using hardware.

5.3 Future Considerations and Application to the Real World

After completing the project, and looking back, there are many improvements and future considerations that could be put forth. For example to make the project more complete and improve the self balancing robot, a line following aspect can be added. This addition was intended by the group, however, due to complications and running out of time, this was not possible. Implementing such aspect would allow the robot to follow a line, while balancing itself upright, without any secondary aid. This could be achieved using IR sensors which would determine what path to follow.

In regards to the working robot itself, it may be advantageous to implement a proper battery system that is less faulty and prone to failure/running out. This would eliminate the troubles that came with the cheap batteries we had. More so, it would be ideal to reprint the custom printed 3D structure with braces to hold in each component. We had used 2 way tape, due to the fact we didn't want to make anything permanent because the kit had to be returned at the end of the semester. Ensuring that each component was secure and wouldn't move would benefit greatly, especially as the constant movement of the absolute orientation sensor kept affecting our algorithm parameters. In addition to a proper battery system, secure and reliable connections would be advantageous. Many times the connections would come loose, as because the components were all positioned in close proximity to either the structure or other components, it proved to be difficult to reattach the connections. A more permanent approach, such as soldering the connections may be one solution to fix this. Again, because these kits had

to be returned, this was not an option for us, however in future iterations, if this is not a factor, it would prove to be helpful.

This project can be applied to the real world as there are various example that imbibe this idea and methods. Examples are: the Segway robots which are used for mobility, they can basically be used to transport people from one point to another. Another example will be the self-balancing scooter, which is used for mobility as well but it is more like a personal scooter that is quite portable unlike the Segway robot.

Appendix A: Project Planning Steps

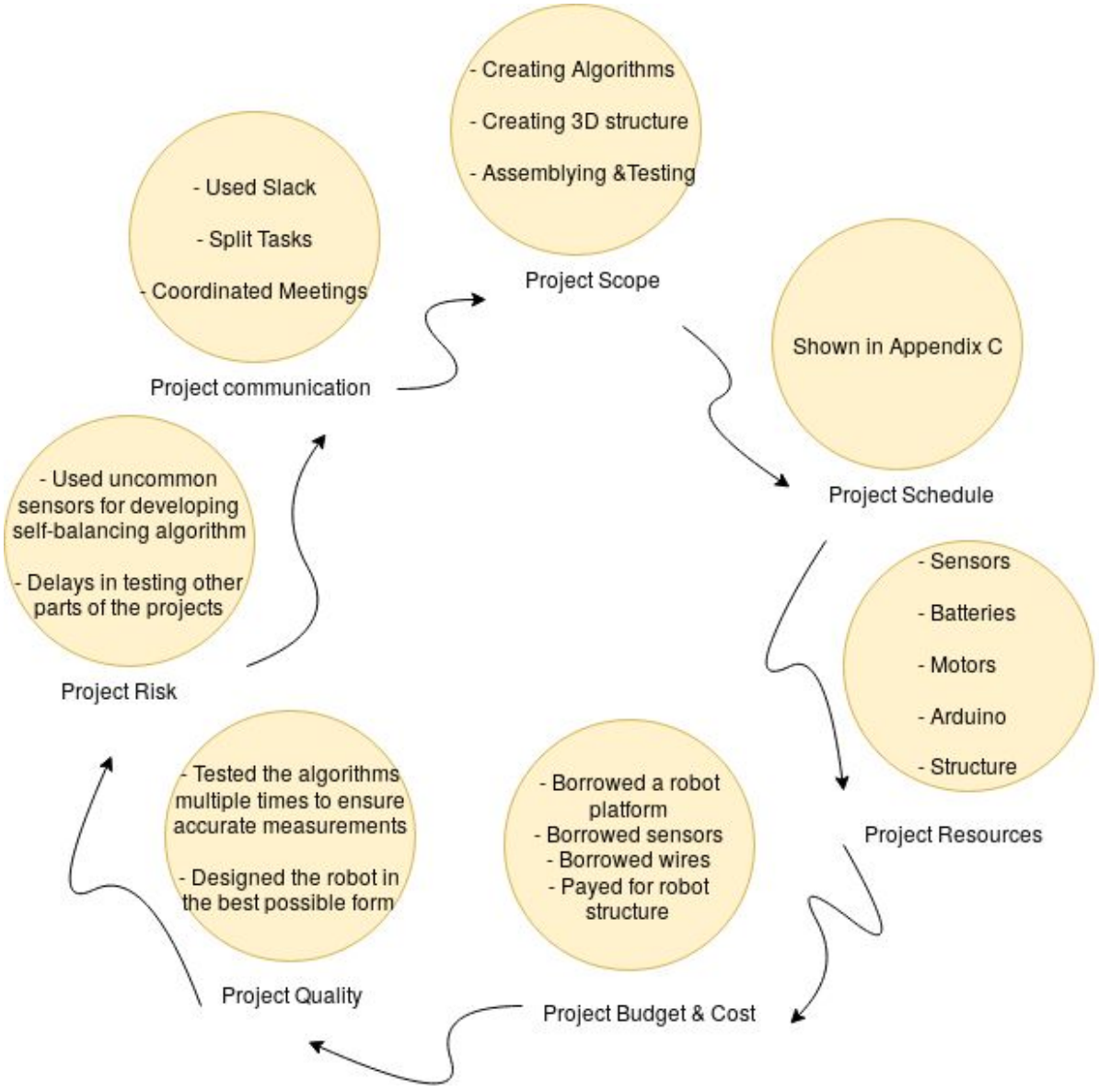


Figure 8: Project Planning diagram

Appendix B: Control Activities

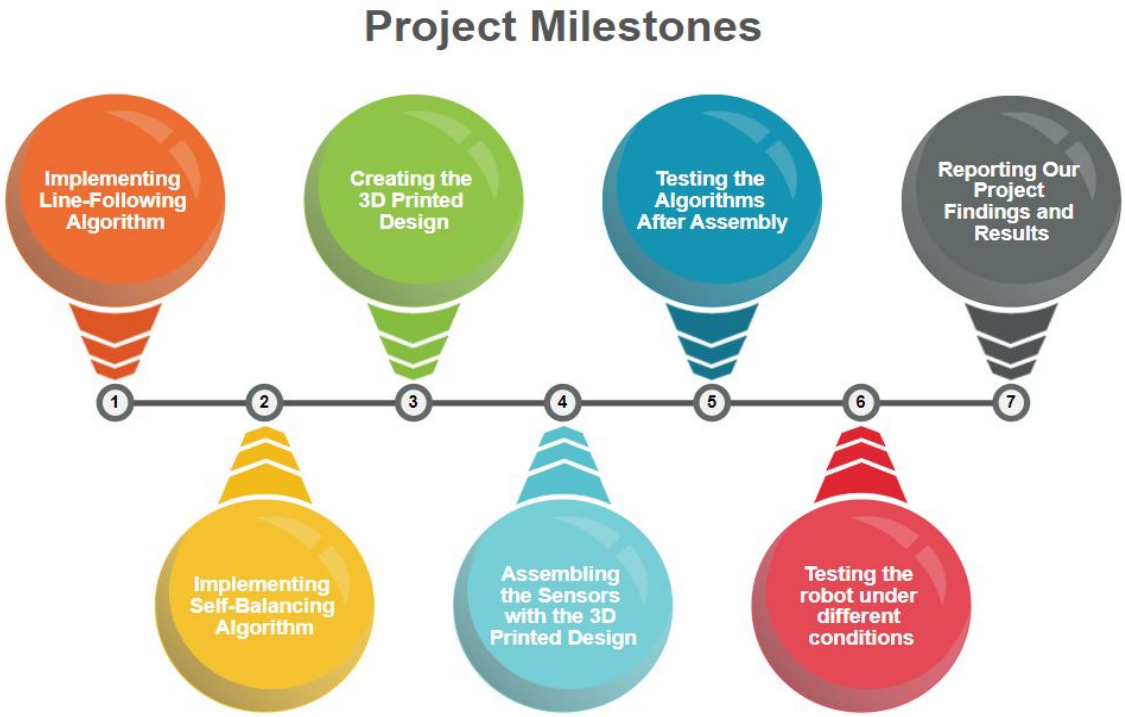


Figure 9: Project Milestones

Appendix C: Minutes of weekly meetings

Table 4: Minutes of weekly meetings

Week	Minutes	Details	Progress
Jan 11	60	-Formed Group and exchanged contact details.	N/A
Jan 14 - 18th	180	-Got Carl5 robot kit and assembled it. Researched various sensors.	-Assembled original Carl5 robot kit.
Jan 21 - 26th	180	-Familiarize ourselves with V-Rep.	-Determined that we would be using physical robot as

			platform.
Jan 28 - Feb 1st	240	-Researched components. Met with Graham Earthly and arranged to get additional hardware. -Project Proposal was written.	-Components needed were acquired. -Project Proposal completed.
Feb 4 - 8th	180	-Testing of components to ensure they were functional. - Begin to research existing libraries and algorithms. - Develop Framework for self balancing algorithms . - Develop line-following algorithm and test it.	-Components were determined to either be functional or non-functional. Non functional parts were replaced. Line-following algorithm was tested and worked.
Feb 11 - 15th	180	-Continue writing framework. -Measurements of original parts so that a new structure can be made.	-Progress in framework development. -Obtained measurements for modeling.
Feb 18 - 22nd	480 +	-Designed and printed 3d structure over reading week.	-3D structure was developed.
Feb 25 - Mar 1st	240	-Assembly of robot with new structure. -Project progress report was written.	-Final assembly of prototype robot. - Progress report completed.
Mar 4 - 8th	240	- Obtain PID values. - Work on algorithms.	-Progress towards software development.
Mar 11 - 15th	240	- Troubleshooting of hardware. - Work on PID values and algorithms.	-Progress on both hardware and software aspects.
Mar 18 - 22nd	180	- Began to create presentation slides. - Work on PID values and algorithms.	-Progress towards software development. -Presentation slides

			created.
Mar 25 - 29th	360 +	- Rehearse and practice presentation - Peer Review forms completed. - Begin writing final report. - Work on PID values and algorithms.	-Presentation of Project was made. -Progress towards software development.
Apr 1 - 5th	360 +	- Prepared for final demo. - Work on final report. -Remaining work on PID and algorithms	-Final Demo made. -Progress on final report. -Wrapped up software aspects of project.
Apr 8 - 9th	360 +	- Work on final report.	Final Report completed and submitted.

Appendix D: Project flowcharts and diagrams

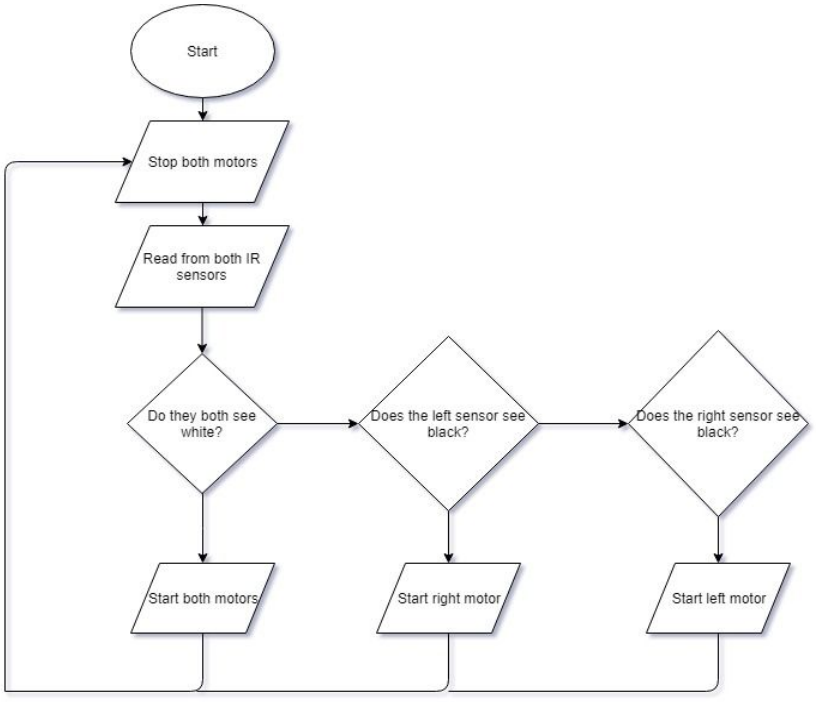


Figure 10: The line following algorithm in a logic chart

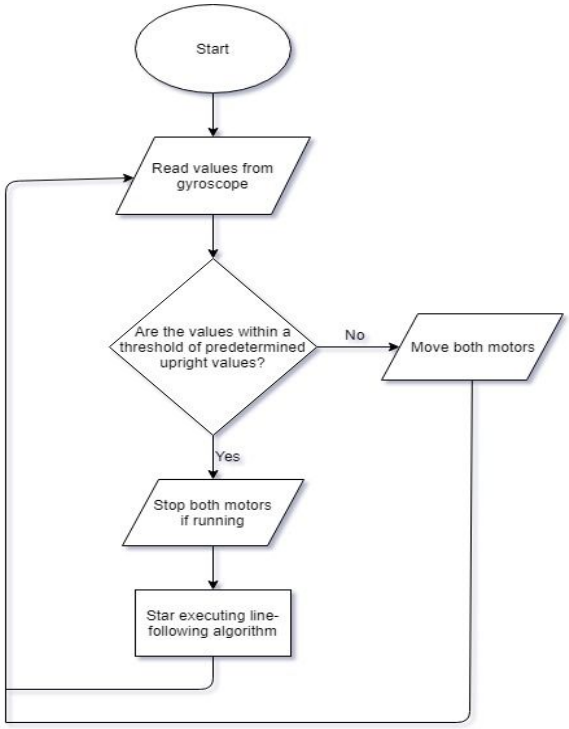


Figure 11: The self-balancing algorithm in a logic chart

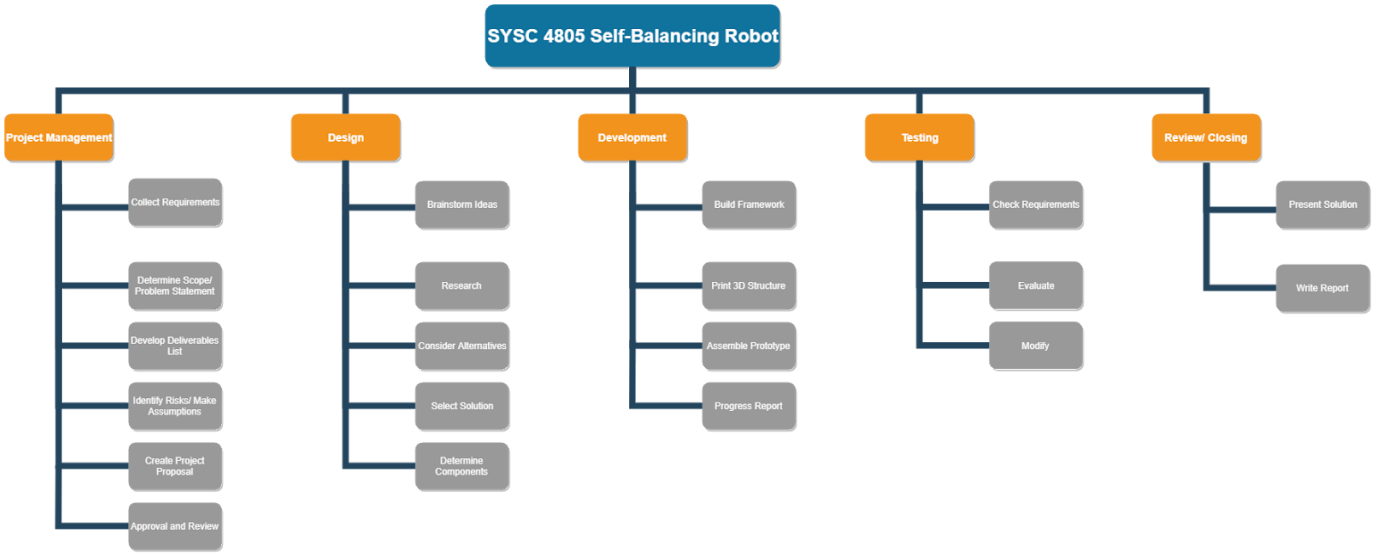


Figure 12: The work breakdown structure of the self-balancing robot project

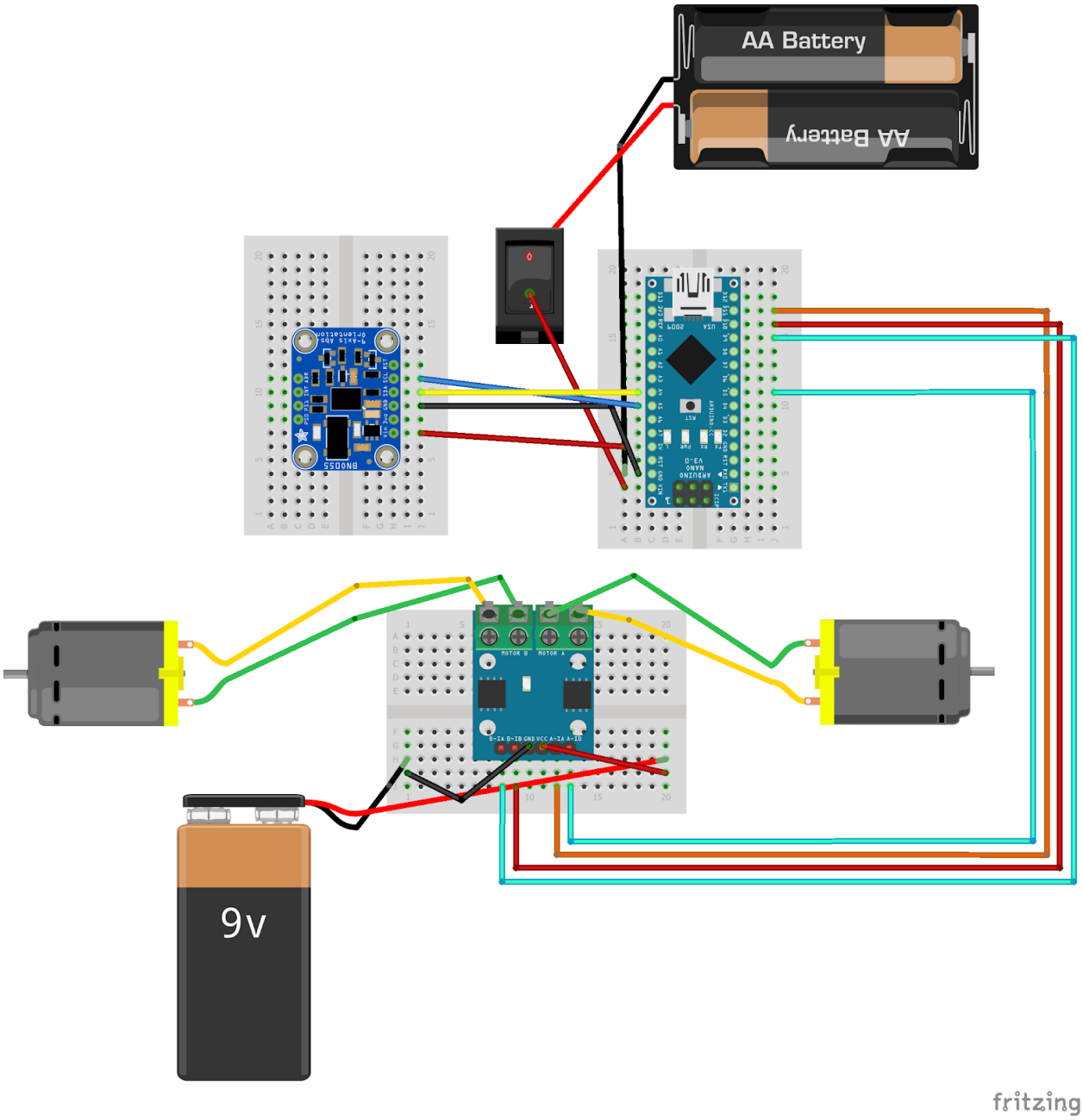


Figure 13: Wiring diagram for Self Balancing Robot

Appendix E: Source Code

```
/*INPUTS*/
#define LS  3
#define RS  6


/*OUTPUTS*/
#define LM1 12
#define LM2 9
#define RM1 8
#define RM2 5

void setup()
{
  pinMode(LS, INPUT);
  pinMode(RS, INPUT);
  pinMode(LM1, OUTPUT);
  pinMode(LM2, OUTPUT);
  pinMode(RM1, OUTPUT);
  pinMode(RM2, OUTPUT);

}

void loop(){
if((digitalRead(LS)==LOW) && (digitalRead(RS)==LOW))
{
    MoveForward();
  }

  if((digitalRead(LS)==HIGH) && (digitalRead(RS)==HIGH))
  {
     Stop();
    }
  if((digitalRead(LS)==LOW) && (digitalRead(RS)==HIGH))
  {
    TurnLeft();
  }
  if((digitalRead(LS)==HIGH) && (digitalRead(RS)==LOW))
  {
    TurnRight();
  }
}



void MoveForward()
{
    analogWrite(LM1, 40);
    analogWrite(LM2, 0);
    analogWrite(RM1, 40);
    analogWrite(RM2, 0);
    delay(10);
}
```

```
void TurnRight()
{
    analogWrite(LM1, 0);
    analogWrite(LM2, 50);
    analogWrite(RM1, 50);
    analogWrite(RM2, 0);
    delay(20);
}

void TurnLeft()
{
    analogWrite(LM1, 50);
    analogWrite(LM2, 0);
    analogWrite(RM1, 0);
    analogWrite(RM2, 50);
    delay(20);
}

void Stop()
{
    analogWrite(LM1, 0);
    analogWrite(LM2, 0);
    analogWrite(RM1, 0);
    analogWrite(RM2, 0);
    delay(10);
}
```

Figure 14: The arduino code for the line following algorithm

Appendix F: CAD Design

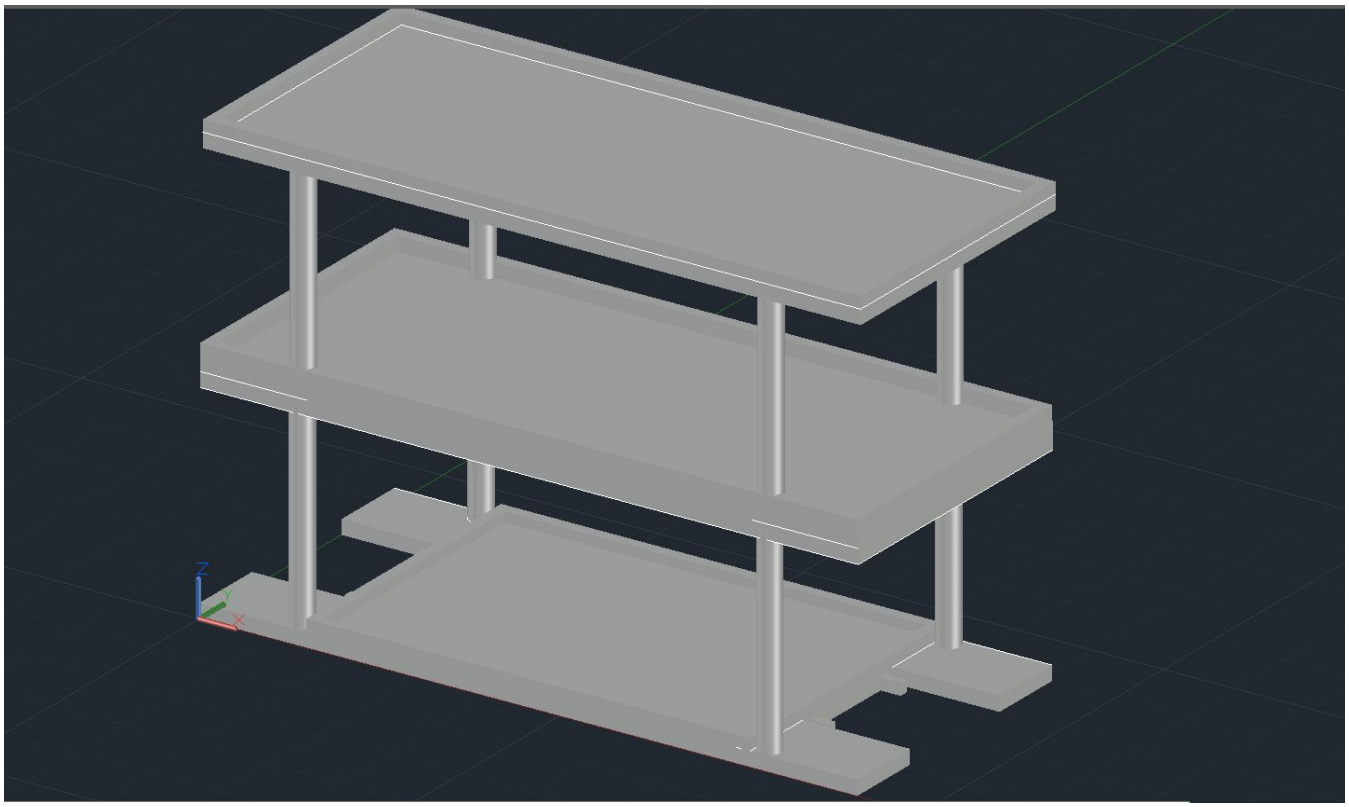


Figure 15: Computer Aided Design Drawing of 3D Printed Structure

References

[1] "S15: Self-Balancing Robot - Embedded Systems Learning Academy", *Socialledge.com*, 2019. [Online]. Available: http://socialledge.com/sjsu/index.php/S15:_Self-Balancing_Robot. [Accessed: 01- Feb- 2019].

[2] "Tandyonline.com. (2019). *Adafruit 9-DOF Absolute Orientation IMU Fusion Breakout - BNO055*. [Online] Available at: https://www.tandyonline.com/adafruit-9-dof-absolute-orientation-imu-fusion-breakout-bno055.html [Accessed 2 Apr. 2019].

[3] Nooraziz.com. (2019). *Arduino Nano V3.0 | Arduino Robots & electronics component in kabul afghanistan*. [online] Available at: http://www.nooraziz.com/product/arduino-nano-v3-0/ [Accessed 2 Apr. 2019].

[4] L. Contr, "L9110S H-bridge Dual DC Stepper Motor Driver Contr | Arduino Robots & electronics component in kabul afghanistan", *Nooraziz.com*, 2019. [Online]. Available: http://www.nooraziz.com/product/l9110s-h-bridge-dual-dc-stepper-motor-driver-contr/. [Accessed: 01- Feb- 2019].

[5] "Rise time, settling time, and other step-response characteristics - MATLAB stepinfo", *Mathworks.com*, 2019. [Online]. Available: https://www.mathworks.com/help/control/ref/stepinfo.html. [Accessed: 02- Apr- 2019].

[6]"PID Tutorial - PID Basics", *Thorlabs.com*, 2019. [Online]. Available: https://www.thorlabs.com/tutorials.cfm?tabID=5dfca308-d07e-46c9-baa0-4defc5c40c3e. [Accessed: 02- Apr- 2019].