

Robert Gordon University



Bachelor of Science in Computer Science
Honours Project Report

Self-Balancing Robot

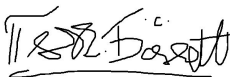
Author:
Petar Bonchev

Supervisor:
Kit-ying Hui

2020

Declaration

I confirm that the work contained in this Honours project report has been composed solely by myself and has not been accepted in any previous application for a degree. All sources of information have been specifically acknowledged and all verbatim extracts are distinguished by quotation marks.

Petar Bonchev: 

May 2020

Abstract

This project focuses on building an autonomous single-wheeled self-balancing humanoid robot made from scratch. This report presents the techniques used to design the robot and using reinforcement learning (RL) and a PID Controller make it self-balancing. In the project was also included a 3D Simulation.

Acknowledgements

Special acknowledgments to my supervisor Mr. Hui, who was guiding me through the whole process of building the robot using FreeCAD. He also helped me choose the appropriate hardware for the project, gave me guidance on electronics, and explained to me the main concepts which would enable me to complete the project. From him, I learned a lot during the development of the project.

Acknowledgments to School of Computing for financially supporting me when it came to purchasing the hardware.

Contents

Declaration	2
Abstract	3
Acknowledgements	4
Contents	5
List of Figures	8

Chapter 1

1.0 Introduction.	10
1.1 Motivation.	11
1.2 Aims and objectives.	11
1.3 Summary and key findings.	11
1.4 Legal, Social, Ethical, Professional and Security issues.	12
1.5 Structure of thesis.	12

Chapter 2

2.0 Literature Review.	13
2.1 PID Controller.	13
2.1.1 Proportional action	14
2.1.2 Integral action.	15
2.1.3 Derivative action.	16
2.1.4 Implementation of PID.	16
2.1.5 Summary	17
2.2 Reinforcement Learning	17
2.2.1 Rewards.	18
2.2.2 States and state-action pairs	18
2.2.3 Agent's Policy.19
2.2.4 Agent's Policy.	20

2.3 Self-balancing robot using PID	21
2.3.1 Design and control of a self-balancing robot using PID. . .	21
2.3.2 Balancing the robot.	21
2.3.3 Motor & Motor Control Board.	22
2.3.4 Development Board.	22
2.3.5 Filter.	23
2.3.6 Wheel-Synchronization.	23
2.3.7 PID Control.	24
2.4 Self-balancing robot using Reinforcement Learning.	25
2.4.1 Q-Learning.	26
2.4.2 Markov Decision Process.	27

Chapter 3

3.0 Hardware.	28
3.1 Setup	29
3.1.1 Raspberry Pi	29
3.1.2 Motor.	30
3.1.3 Sensor.	30
3.1.4 Batteries.	31
3.2 Modifications.	32
3.3 Issues	33

Chapter 4

4.0 Design.	35
4.1 First Prototype.	36
4.2 Second Prototype.	38
4.3 Issues.	40

Chapter 5

5.0 Implementation.	42
5.1 FreeCAD.	42
5.2 Python.	44
5.2.1 PID implementation	44

5.2.2 Reinforcement Learning.	45
5.2.3 Filters.	47
5.3 C# and Unity.	49

Chapter 6

6.0 Testing and Results.	50
6.1 Sensors.	50
6.2 Physical Simulation.	51
6.2.1 PID values.	52
6.3 Software Simulation.	53
6.4 Results.	54
6.4.1 Sensor.	54
6.4.2 Prototypes.	54

Chapter 7

7.0 Evaluation.	56
7.1 Self-Balancing.	56
7.2 Legal, Social, ethical and professional issues.	58
7.3 Improvements.	58

Chapter 8

7.0 Conclusion	60
7.1 Critical Appraisal.	61

References.	62
Source code.	64
Ethics Form.	64

List of Figures

Figure 1: PID controlled system.....	13
Figure 2: PID Formula.....	14
Figure 3: Proportional response action output.....	14
Figure 4: Integral response action zero point example.....	15
Figure 5: Integral calculation example.....	15
Figure 6: PID pseudocode.....	16
Figure 7: Reinforcement Learning structure.....	17
Figure 8: Reinforcement Learning state action example	18
Figure 9: Reinforcement Learning agent and environment example.....	19
Figure 10: Self-balancing robot using PID design.....	21
Figure 11: Complementary Filter and PID controller.....	24
Figure 12: Self-balancing robot using Reinforcement Learning design.....	25
Figure 13: Self-balancing robot using Reinforcement Learning hardware.....	25
Figure 14: Q-Learning iteration.....	26
Figure 15: Q-Learning pseudocode.....	27
Figure 16: Raspberry PI 4 model B.....	29
Figure 17: Raspberry PI pins	29
Figure 18: Motor connected to PI.....	30
Figure 19: MPU-6050 sensor	30
Figure 20: MPU-6050 connected to Pi	31
Figure 21: Batteries and battery holder.....	31
Figure 22: UBEC regulator	32
Figure 23: Battery charger	32
Figure 24: Modifications made.....	33
Figure 24-1: Wi-Fi command 1.....	33
Figure 24-2: Wi-Fi command 2.....	33
Figure 24-3: Wi-Fi command 3.....	34
Figure 24-4: Wi-Fi command 4.....	34
Figure 24-5: Wi-Fi command 5.....	34
Figure 25: Robert Gordon Robot	35
Figure 26: First prototype	36
Figure 27: Leg holder	36
Figure 28: Motor Holder	37

Figure 29: Rod holder	37
Figure 30: Body of Robot	37
Figure 31: Prototype 2	38
Figure 32: Motor holder improved	38
Figure 33: Leg holder improved	38
Figure 34: Rod holder improved	38
Figure 35: Second prototype first iteration side.....	39
Figure 36: Second prototype first iteration front.....	39
Figure 37: Second prototype first iteration side 2	39
Figure 38: FreeCAD part example	42
Figure 39: FreeCAD legPart structure	42
Figure 40: FreeCAD part adjustments 1	43
Figure 41: FreeCAD part adjustments 2	43
Figure 42: PID code snippet	45
Figure 43: Duty cycle example	45
Figure 44: Reinforcement Learning example	46
Figure 45: Complementary filter example	47
Figure 46: Complementary filter code.....	48
Figure 47: Filters structure.....	48
Figure 48: Unity Robot design	49
Figure 49: Unity Robot balancing	53
Figure 50: Unity Robot falling.....	53
Figure 51: Unity Robot regaining balance.....	53
Figure 52: Prototype 1 - balancing	55
Figure 53: Prototype 1 - pushed	55
Figure 54: Prototype 1 - regaining balance.....	55
Figure 55: Prototype 2 - final.....	55
Figure 56: Self-Balancing comparison	57

Chapter 1

1.0 Introduction

A complex challenge that could test my knowledge obtained in university that has captured my attention is building a robot. A combination of both hardware and software elements would converge to arrive at a functioning unit. Moreover, a robot that is balancing itself presented a further challenge.

This project focuses on building a single-wheeled self-balancing robot with a humanoid body form printed using a 3D printer. The goal in terms of design was for the robot to resemble the Robert Gordon University robot, also referred to as 'Glitch'. Using specific hardware including motors which would move accordingly depending on whether the robot is in a balanced position or in a falling state.

Maintaining self-balance is a challenge and it has to be implemented using a PID controller and then with Reinforcement Learning. Multiple examples of self-balancing single-wheeled robots can be found on the internet using different mathematical and physics equations to maintain the stability of the robot. However, none of the online designs were copied but only used as examples. In this project a brand new design is created. Software simulation had to be created as well using the same software approaches, simulating how the robot behaves and balances in a 3D simulated world.

A commercially available self-balancing robot could be of huge benefit to humans. The efficiency of the robot could help humans with different simple tasks. For example, it could be used in the household for cleaning or in the assistance of elderly or disabled people. The project will explore and display how a self-balancing robot will behave in our everyday environment with all safety and security measures taken into account.

1.1 Motivation

The group that would be interested would be all companies that are producing Robots and/or Robotics related software. The companies could benefit from the project by using the already built Robot to build on top of the project per se by, for instance, expanding features so the robot can, for instance, detect dangers, help with simple tasks, entertain, explore danger for humans places, etc. The project offers an initial schematic and guidelines including the code written to run it that can be repurposed in any of its elements.

Using the robot made in this project, a company can also use it to present in front of their clients how the robot behaves in a friendly safe for people way so the clients can become more familiar with the core of the Robotics involved. Other use cases could be in education or product marketing. The robot can be used to demonstrate the potential and abilities of the technology to students or to customers for future sales. Moreover, it can show investors the potential of the robotics industry.

1.2 Aims and objectives

Objective 0: Build the robot using a 3D printer and Raspberry PI

Objective 1: Study different algorithms and formulas that would be suitable for self-balancing the robot by identifying their strengths and weaknesses, and deciding which ones are most suitable for the project.

Objective 2: Implement the chosen technologies to self-balance the robot

Objective 3: Implement the chosen technologies to make the robot move

Objective 4: Conduct multiple various tests and trials to evaluate the results of the self-balancing and moving part of the project and its capabilities as part of the prototype.

Objective 5: Create a refined self-balancing Robot that moves

Objective 6: Research reinforcement learning for self-balancing robots and other relevant literature.

Objective 7: Implement reinforcement learning and compare the result to the other technique used to move the robot

1.3 Summary of key findings

1. Computer-aided design (CAD) modeling - FreeCAD software
2. 3D printing - to create the parts
3. Electronic knowledge - to put the parts together
4. Libraries - RPi.GPIO, PiGPIO, smbus
5. To make the robot self-balancing I could use PID since it is widely used as a control system to maintain a setting.
6. To make the robot self-balancing and stability I could also use reinforcement learning since it makes the robot learn by itself and thus maintain balancing and stability:

1.4 Legal, Social, Ethical, Professional and Security issues

Design is of paramount importance when it comes to robotics. How you design your robot and the hardware you are using and their safety because if these things are not taken into account fire, explosion, or electricity surge could occur. When designing a robot and writing the code for it, all testing should be done safely because otherwise the security of the people surrounding the robot could be compromised.

1.5 Structure of thesis

Chapter 1 - Introduction - an introduction to the projects' motivation, aims, objectives, key findings and issues.

Chapter 2 - Literature Review - A broad literature review on PID controllers, Reinforcement Learning, and self-balancing robots.

Chapter 3 - Hardware - What hardware was used, why the particular hardware was selected, and how it was put together.

Chapter 4 - Design - Structure of the designs of the robot and details about the specific parts

Chapter 5 - Implementation - specific implementation details about each of the software-hardware components used.

Chapter 6 - Tests and Results - Details about all the self-balancing tests implemented and their results

Chapter 7 - Evaluation - Comparing the results using two different prototypes and their optimal setup and other related information

Chapter 2

2.0 Literature Review

The usage of PID controller and Reinforcement Learning to maintain control of a robot's balance seems like a great approach to tackle the issue this project aims to resolve and has shown promising results in the past. This chapter will review the literature surrounding these two different techniques to self-balance a robot, namely, using a PID controller and reinforcement learning. The review also describes what the two techniques are and how they were used in the past, in other projects whose aim was the same or similar to this project, to successfully self-balance a robot.

2.1 PID Controller

Proportional-integral-derivative (PID) control is a simple three-term controller that is used to control 95% of the closed-loop industry processes; and has been universally accepted in industrial control. The PID control consists of three basic coefficients: Proportional (P), Integral (I), and Derivative (D) which are completely different from one another to get an optimal response and their combined output is used in the form of a control algorithm to control 95% of closed-loop industrial processes such as robotics (Zhong, 2006). From Figure 1 below, which illustrates how a PID controller system behaves (Paz, 2001), $R(s)$ is the setpoint which stands for the value that we want the process $G(s)$ to be. $E(s)$ is a tracking error function which equals the setpoint $R(s)$ minus the control error $Y(s)$. Hence, the tracking error function output is passed to the PID controller which consists of three-terms. The three-terms (P) K_p , (I) K_i/s , (D) $K_d s$, which all work independently from one another, when summed up together output a signal $U(s)$ that will produce output equal to the

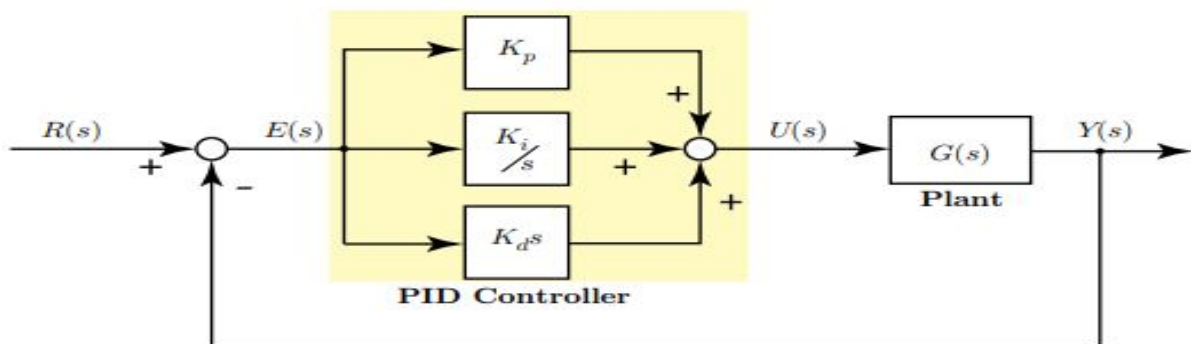


Figure 1 (PID controlled system)

setpoint, in other words, will try to get the signal $U(s)$ to zero. The process $G(s)$ is the Plant of the system which could be, for instance, a DC motor.

The output $U(s)$ can be derived from the formula below (Figure 2) (Zhong, 2006). The control signal U from the controller, which is the signal that is being sent to the plant, is equal to the proportional gain K_p times the magnitude of the error, plus the integral gain K_i/s times the integral of the error, plus the derivative gain $K_d s$ times the derivative of the error (Paz, 2001)(Zhong, 2006).

$$u = K_p e + K_i \int e dt + K_d \frac{de}{dt}$$

Figure 2

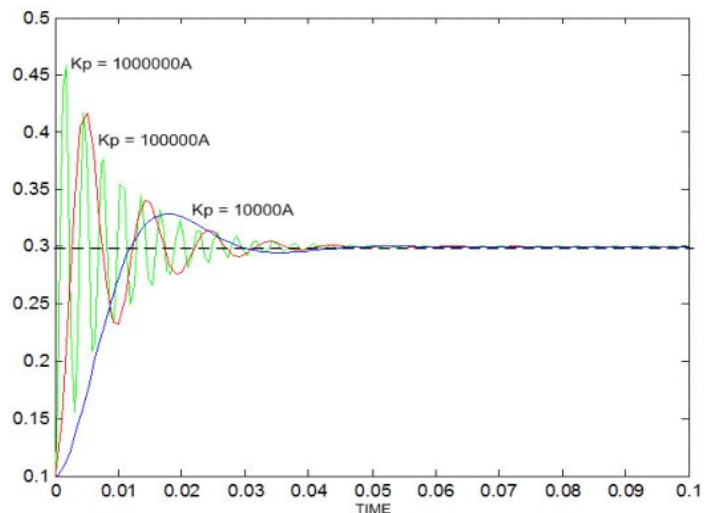
However, even though they work independently, their impact on the closed-loop depends on all three of them (Graham C Goodwin, 2001). In other words, removing either one of them would result in damaging the accuracy of the closed-loop process output.

2.1.1 Proportional response action

The proportional gain K_p is a setting that we tune to determine the ratio of the output response to the error signal which is the desired performance. For example, given an error term that has a magnitude of 8, a proportional gain of 4 would produce a proportional response of 32.

Figure 3

In other words, the proportional action simply multiplies the error by the proportional gain K_p to get the controller output (Paz, 2001). However, if the proportional gain is too massive, the process variable will begin to oscillate, thus making the proportional action output highly unstable (see Figure 3).



The contribution of the proportional action depends on the instantaneous value of the control

error $Y(s)$ (Graham C Goodwin, 2001). A proportional controller on its own can control any stable plant, but it provides limited performance and nonzero steady-state errors. Since the proportional action entirely depends on the error

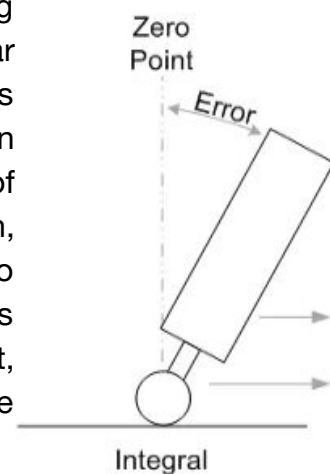
signal, it can be considered as an action that aims to solve the errors in the past or errors that have already happened.

2.1.2 Integral response action

The integral component is based on the current angle difference from Point Zero (the setpoint $R(s)$) multiplied by the I gain, which is accumulated over time (Graham C Goodwin, 2001). Since the steady-state error is the final difference between the process variable and setpoint, the result is that even a small error term will cause the integral component to increase slowly over time unless the error is zero, therefore the aim is to drive the steady-state error to zero (see Figure 4 and 5). The output of the controller is proportional to the accumulated error, which implies that the integral response action is a slow reaction control mode.

The integral mode plays a fundamental role in achieving perfect plant inversion at $\omega = 0$ (ω stands for Angular Frequency) when an event causes the plant to change its steady-state position. Using the integral response action this forces the steady-state error to zero in the presence of a step reference and disturbance. (Graham C Goodwin, 2001). For instance, keeping the steady-state error to zero will help a self-balancing robot keep balance when it starts losing balance or changes its body angle due to movement, maneuvering, or an external occasion such as if someone or something pushes the robot.

Figure 4



The integral mode, viewed in isolation, has two major shortcomings, namely, its pole at the origin is detrimental to loop stability, a stability issue that can arise is sometimes referred to as integral windup, and it also gives rise to the undesirable effect (Graham C Goodwin, 2001). An integral windup could occur when the integral term reaches a large positive or negative value and thus the controller loses the ability to regulate the process. Hence, since the integral action takes into account the error instantaneously, it can be considered as an action that aims to resolve a problem at the present time.

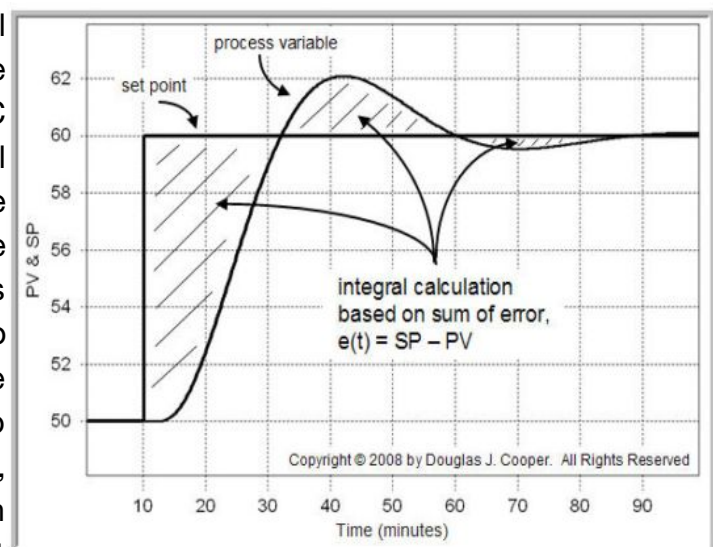


Figure 5 (PV=process variable, SP=Set Point)

2.1.3 Derivative response action

In general, the derivative component causes the output to decrease if the process variable is increasing quickly and is commonly referred to as a predictive mode because of its dependence on the error trend (Graham C Goodwin, 2001). Basically, the smaller the derivative time constant is, the larger the frequency range over which the filtered derivative approximates the exact derivative with equality in the limit. In other words, the rate of change of the derivative response is proportional to the process variable, increasing the derivative time K_d s parameter will cause the control system to respond more firmly to changes in the error term and will increase the speed of the overall control system response

Hence, most practical control systems use very small derivative time K_d s due to the reason that the derivative response is highly sensitive to noise in the process variable signal (Knospe, 2006). In case the sensor feedback signal is noisy or the control loop rate is too slow, the derivative response can make the control system unstable, thus the derivative action acts on the rate of change of the control error.

The main limitation of the derivative mode, when viewed in isolation, is its tendency to yield large control signals in response to high-frequency control errors, such as errors induced by setpoint changes or measurement noise.

2.1.4 Implementation of PID

As seen in Figure 6, the pseudo-code for a PID controller is not large in size. Initially, all three values have to be set up, namely - (P) K_p , (I) K_i , (D) K_d , also the *error_prior* and *integral* are set to a default of zero.

The implementation happens within a closed-loop - a while statement. With each iteration the (P), (I), (D) response actions are calculated using the PID formula (see Figure 2) and the output updated. Finally, an interval of iteration time is set, for example - 1.0s.

Figure 6 (Pseudocode for PID)

```
error_prior = 0
integral = 0
KP = Some value
KI = Some value
KD = Some value

while(1){
    error = desired_value - actual_value
    integral = integral + (error*iteration_time)
    derivative = (error - error_prior)/iteration_time
    output = KP*error + KI*integral + KD*derivative
    error_prior = error
    sleep(iteration_time)
}
```


2.1.5 Summary

Mostly used in closed-loop processes such as robotics, the PID control algorithm is a simple, yet quite a powerful algorithm that is widely used in the industry. The algorithm has sufficient flexibility to yield excellent results in a huge variety of processes and has been one of the fundamental reasons for continued use throughout the years.

The proportional action can be considered as an action that fixes an error that has happened in the past. The Integral action aims to solve the errors that are happening at the present, and the derivative action predicts if any errors are about to happen and if they do, take the corresponding action.

2.2 Reinforcement Learning

Reinforcement Learning (RL) is a computational active agent-based framework whose purpose is to make an agent learn different behaviors, using reward signals to determine if they are going well or not (Doya, 2007) (see Figure 7). RL is not supervised as it does not rely only on a set of particular training data, also, it is not unsupervised due to the reason that there is a specific reward that the agent will aim to maximize (Welling, 2012), hence RL is considered to be just in between supervised learning and unsupervised learning. RL has an excellent wide usage in Robotics (Kober, 2013), consequently, using RL for a self-balancing robot w

Reinforcement learning deals with learning issues in a sequential decision-making approach in which limited feedback is found (Welling, 2012). Therefore, RL finds the correct actions, which are set initially, to take in different situations to achieve its overall goal.

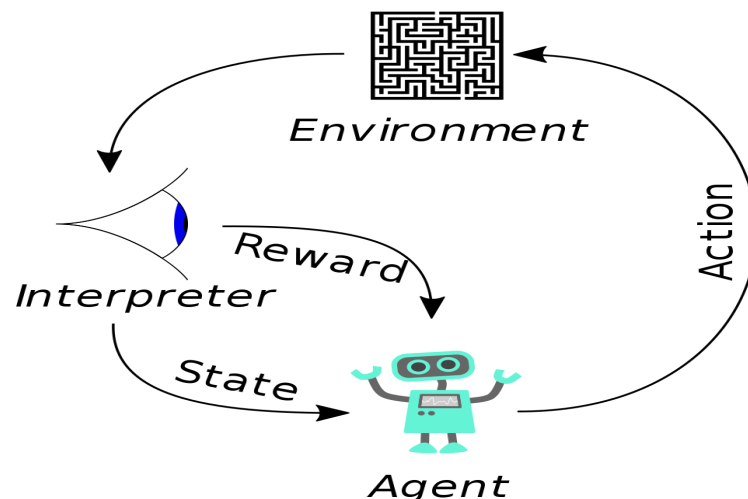


Figure 7

2.2.1 Rewards

The specific reward function is objective feedback from the environment which defines what is objectively good and bad for the agent (Sutton & Barto, 1998). Moreover, the rewards are unalterable by the agent and could be an integer or scalar variable associated with some states or state-action pairs. This association defines the goal of the agent in any given situation. The actions that are taken influence the state of the world which determines its reward (see Figure 8) and the agent's sole objective is to maximize total net long-term rewards. In other words, the reward function specifies the states associated with primary reinforcement.

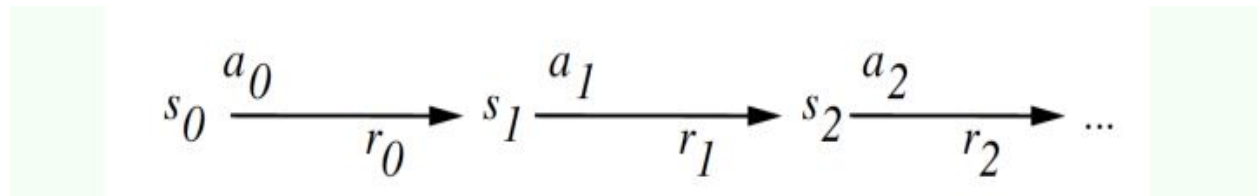


Figure 8

For instance, a reward of 1 might be identified with the state of balancing the robot at a specific angle or range, whereas a reward of 0 is associated with all other states of the robot that do not correspond to the desired angle/goal, which might result in the robot losing its balance. Thus, if the specific goal angle corresponds to the angle required to balance the robot, the robot will not lose balance because it would be aiming at getting and increasing its total net long-term rewards. Hence, it is of paramount importance to set the goal accurately otherwise the robot would not keep its balance as it should.

Another basic instance of RL, in a chess game, a reward of 1 could be associated with the state of having captured all of the enemy's pieces or capturing the opponent's king, while a reward of 0 could be identified with all other states of the game.

2.2.2 States and state-action pairs

A state is any factor that the agent might take into consideration in making that decision, whereas an action is any decision an agent might need to learn how to make. The state on which an action is predicated may include an environmental model (Doya, 2007), thus representing past states of the actor's environment (as seen in Figure 8).

Noticeably, all representations would be considered part of the agent's state at the time it decides on an action. It seems to be at least in principle, neutral on the issue of what knowledge is and how it determines action.

2.2.3 Agent's policy

The ultimate goal of reinforcement learning is to learn a policy that returns an action to take given a state (Doya, 2007). The agent's policy, which is set up in the policy function, is the mapping from possible states to possible actions, in other words, the policy can be considered as an agent's strategy to achieve particular tasks.

To form a good policy, we need to know the value of any given state and calculate which actions would be the most efficient to get to the goal state. Therefore, using an already predefined policy function, by learning a value function which is the sum of rewards from the current state to some terminal state (Doya, 2007). The mapping could be either in the form of a look-up table or alternatively, the agent may rely on a computed policy. A computed policy, for instance, may involve a search through an ever-changing tree of values, using a model of the environment to look for action sequences that produce the greatest value.

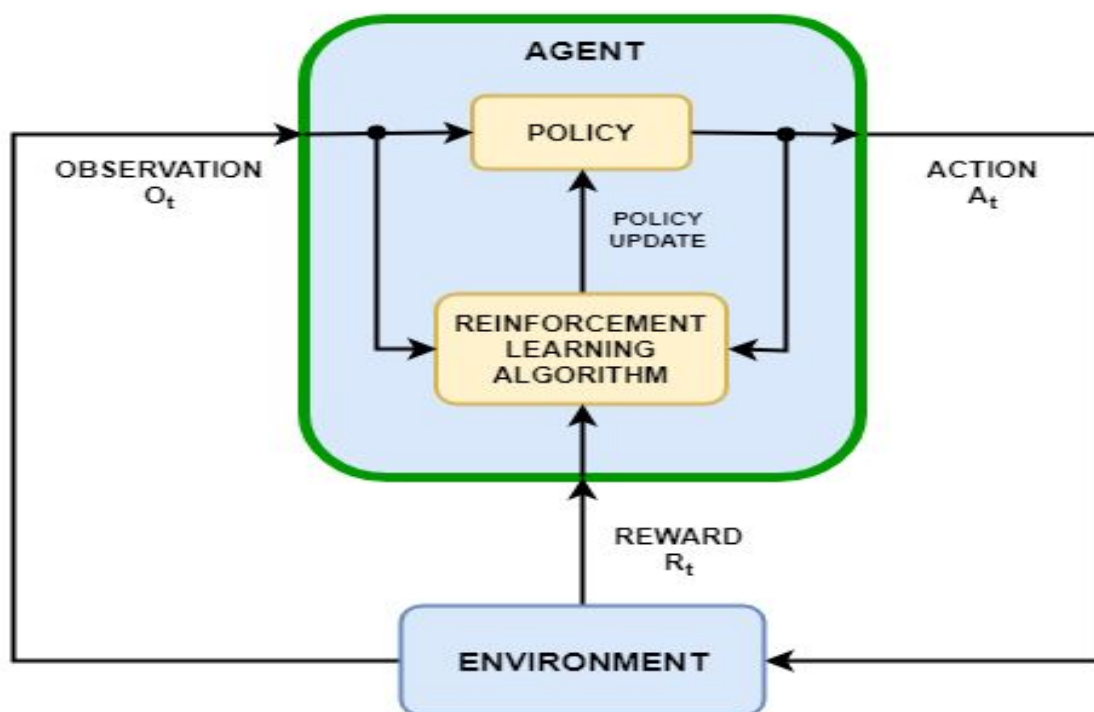


Figure 9 (The Agent, the policy, and the RL algorithm)

For instance, a self-balancing robot moves across a university room and its goal is to get to a particular point within that room, provided that the robot reaches that goal it will receive a specific reward. In this case, the room is the environment, the robot's current position is a state and the policy is what the agent will be doing to achieve this task. Many different policies can be used to accomplish this particular

task, however, not all of them are efficient, thus a problem arises which is how efficient is the agent's policy function.

An inefficient policy could be to simply let the robot roam around continuously in expectation to accidentally wind up in the correct place. Comparingly, a more efficient way to get to the goal destination could be to actually plan/calculate the route beforehand and go straight to the goal.

The policy function is the agent's current mapping from the set of possible states or state-action pairs to its estimates of the net long term reward to be expected after visiting a state/state-action pair and continuing to act accordingly to its current policy thereafter (Doya, 2007). The policy function is continually updated by the agent based on the experienced consequences of its actions. These experienced consequences are ultimately the rewards it receives, but the consequences that most immediately matter to the updating process are the values of the states to which the agent gets in the next few actions. (Doya, 2007).

2.2.4 Summary

Reinforcement learning (RL) is a type of machine learning that determines the action within a specific environment in order to maximize a reward. The problem-based setup of RL captures the primitive features of animal and human behavior, thus creating an adaptive mechanism. An agent can only receive a reward after performing the action, and thus must continue to interact with the environment in order to determine the optimal policy through trial and error.

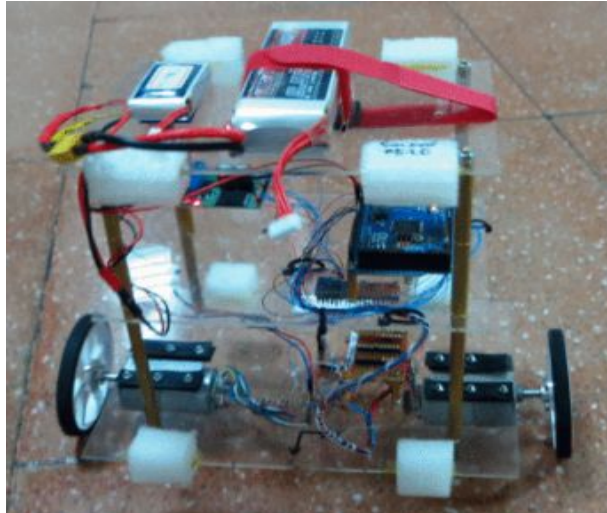
An important part of an agent is the agent's policy which determines how the agent is going to tackle the problem. The policy function is used to map/create a plan of how an agent would solve the problem by analyzing and calculating different sets of possible states and state-action pairs; thus aiming to come up with the most long term rewarding path. A huge variety of policies can be used to solve a problem, however, not all of the policies are efficient enough, depending on the task.

2.3 Self-Balancing Robot using PID

In this section is discussed a project in which PID was used in order to make a robot to maintain self-balance. Some of the techniques involved to make the robot keep balance also include different techniques on top of the PID controller.

2.3.1 Design and control of a self-balancing robot using PID

This project is about the design, construction and control of a two-wheel self-balancing robot that uses a PID controlled system (see Figure 10) (Hau-Shiue Juang & Kai-Yew Lurrr, 2013). The robot is driven by two DC motors and is equipped with an Arduino board, a single-axis gyroscope and a 2-axis accelerometer for attitude determination and a Complementary filter.



2.3.2 Balancing the robot

Figure 10

To balance the robot, the data of the robot's tilting angle and angular velocity are needed in real-time to be calculated (Hau-Shiue Juang & Kai-Yew Lurrr, 2013). Consequently, using MEMS (Micro-electro-mechanical systems) sensors including a gyroscope, an accelerometer, and wheel-angle encoders, all the data can be measured needed for balancing control of the robot.

The gyroscope is the sensor that can measure the angular velocity of the balancing robot, maintain its orientation, and send the data to the development board (Oxford Dictionaries, 2019). The angular velocity from the gyroscope is used to return the angle directly, thus noise from gyroscope measurement will be integrated as well, which noise can be removed by using Kalman filter or Complementary filter (Kalman, 1960; Brookner, 1998). The accelerometer measures the total external acceleration of the balancing robot, which includes the gravitational and motion accelerations (Richard F. Tinder, 2007).

The encoders return the rotation angles of individual motor shafts as digital or analog signals, which are sent to the processor. After conversion based on the gear ratio and wheel radius, the distance traveled can be calculated. (Hau-Shiue Juang & Kai-Yew Lurrr, 2013).

2.3.3 Motor & Motor Control Board

Motor selection for the balancing robot emphasizes torque output instead of velocity because it has to oppose the rotational moment that gravity applies to the robot. Hence, the motors need to provide enough torque to correct the robot's body pose back to a balanced state. In the case of this implemented project, the large currents drawn by the two motors are supplied by the dual-motor control board, which can deliver sufficient continuous output current at a maximum operating voltage of 16 Volts (Hau-Shiue Juang & Kai-Yew Lurrr, 2013).

2.3.4 Development Board

When selecting a development board in order to build a robot, there are a couple of considerations that should be taken into account such as Performance, Input & Output Pins, Open Source, and Power supply, in order to get the most suitable development board.

The self-balancing robot needs as close as possible to real-time response to estimate and correct its tilt angle. Hence, the development board must provide a processing speed that is sufficiently fast to perform the processing tasks, including data acquisition, control computation, and signal output, within the sampling time.

A robot's sensors are deployed to obtain measurements of its motion: a gyroscope and an accelerometer are used to estimate the tilt angle, encoders are used to obtain odometric measurements. Hence, making sure that the amount of I/O pins on the board is enough is of paramount importance, since otherwise there will not be enough space for one or more particular devices to be connected.

To alleviate difficulties in the development of the software for the robot, a user-friendly development environment, useful function libraries, and references are preferred. These requirements are well met by the Arduino development environment which is based on the C language.

For the power supply, the motors need a voltage between 12V to 16V, and the development board needs between 5V to 15V. Consequently, a certain amount of batteries have to be incorporated, depending on the number of voltages the batteries have. For instance, for the robot in this project presented in Figure 10, a

14.8-volt lithium battery was used and for the development board a four-cell (4×1.5 volts) Ni-MH battery pack.

2.3.5 Filter

When using electronics such as gyroscope, a filter is required since a gyroscope measurement contains noise (Kalman, 1960; Brookner, 1998). This will make the integrated data diverge from the correct angle and thus the robot could lose balance. Therefore, using a Complementary filter resolves the issue, since it takes slow-moving signals from the accelerometer and fast-moving signals from a gyroscope and combines them for a more precise overall measurement; thus, the noise is either reduced to zero or eliminated fully (Brookner, 1998).

Alternatively, a more sophisticated type of filtering is Kalman filtering, which is also often called linear quadratic estimation (LQE). LQE is an algorithm that uses a series of measurements observed over time, such as the ones returned from a gyroscope and an accelerometer, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement per se (Kalman, 1960). In this project was used the Complementary filter since it is more simple to implement (Brookner, 1998; Hau-Shiue Juang & Kai-Yew Lurrr, 2013).

2.3.6 Wheel-synchronization

Generally, the same motor control signal is sent to the motors control board should cause all the motors to rotate at the same speed. However, in our everyday environment, there are many reasons for which the motors of the robot could happen to rotate at different velocities under the same input signal, such as defects of the motors, terrain, and hindrance on the ground. Consequently, an approach to synchronize the motors is needed.

To resolve this issue in this project a wheel synchronization controller was designed which consists of a simple Proportional-Integral (PI) controller. The PI controller alternates the signal to the motors so that the difference between the left and right encoders tracks zero, thus resolving this defect and synchronizing the wheels (Brookner, 1998; Hau-Shiue Juang & Kai-Yew Lurrr, 2013).

2.3.7 PID Control

The PID control is chosen in this project because it is efficient, easy to learn and to implement (Hau-Shiue Juang & Kai-Yew Lurr, 2013). The block diagram of the PID controller is as shown in Figure 11, wherein the complementary filter generates estimates of the angular velocity and angle. Instead of differentiating the angle measurement which will amplify noise, differential control is implemented by multiplying angular velocity with the differential gain K_d . Proportional control is given by θ multiplied by the proportional gain K_p , whereas the numerical integration of θ and multiplication by the integral gain K_i yields the integral control.

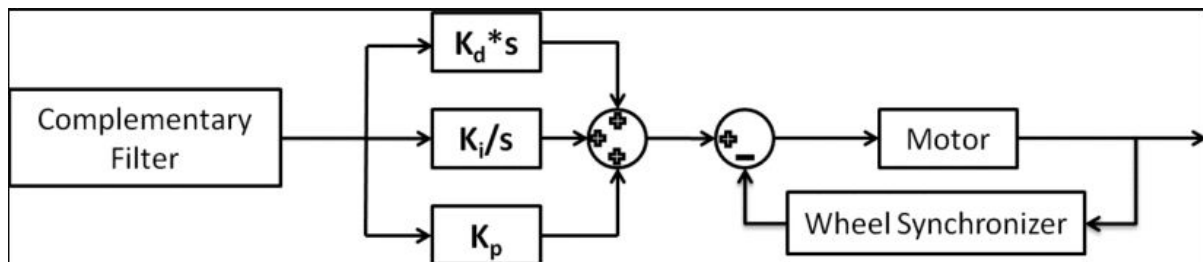


Figure 11

2.4 Self-Balancing Robot using Reinforcement learning

In this section is discussed a project in which was used Reinforcement Learning(RL) in order to make a robot to maintain self-balance. Some of the techniques involved to make the robot keep balance also include different techniques alongside Reinforcement Learning.

2.4.1 Introduction

In this project are introduced the techniques used to create a self-balancing robot. Those techniques are Markov Decision Process for Reinforcement Learning and Q-Learning, which is a type of Reinforcement Learning. Additionally, the Complementary filter approach with a PI-PD control was used alongside Reinforcement Learning to increase the balance accuracy and mitigate any risks that may result in the robot losing its balance (Shih-Yu Chang & Ching-Lung Chang, 2016).

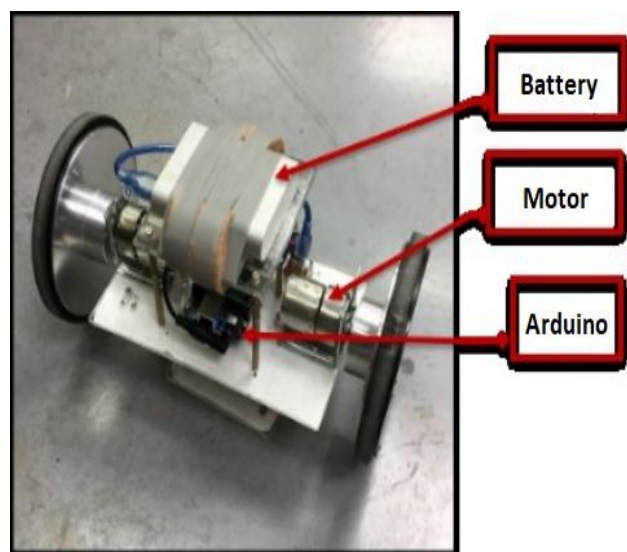


Figure 12

As seen in Figure 12 and Figure 13, the self-balancing robot consists of a battery, two motors, Arduino, motor driver, accelerometer, gyroscope, a body, and two wheels.

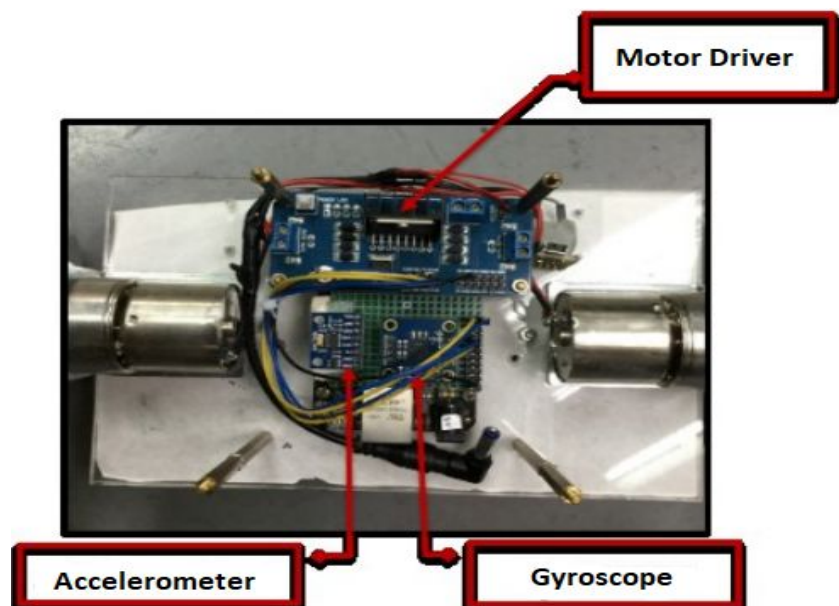


Figure 13

2.4.2 Q-Learning

As mentioned in 2.2 of this document, Reinforcement learning (RL) is a category of machine learning that aims to maximize a reward by trying to determine the action within a specific environment in order to maximize a reward. An important characteristic of reinforcement learning is that the agent can only receive a reward after performing the action, consequently, the robot per se must continue interacting with a specific environment to determine the optimal agent policy through different trial and error. Therefore, the longer an agent, in this project the agent is a robot, interacts with an environment, the higher the chance to determine the optimal policy (Welling, 2012).

Since Q-Learning is a Reinforcement Learning type whose main difference is that the maximum reward for the next state is not necessarily used for updating the Q-values, instead a new action and a new reward are selected using the same policy that determined the initial action (Doya, 2007). In this project, Q-learning has been implemented in order for the robot in Figure 12 & Figure 13 to maintain balance, which gives an example of a simple and efficient way for an agent to learn how to behave in an optimal manner to maintain balance when moving and manoeuvring. It works by successively improving its evaluations of the quality of particular actions at specific states (Shih-Yu Chang & Ching-Lung Chang, 2016).

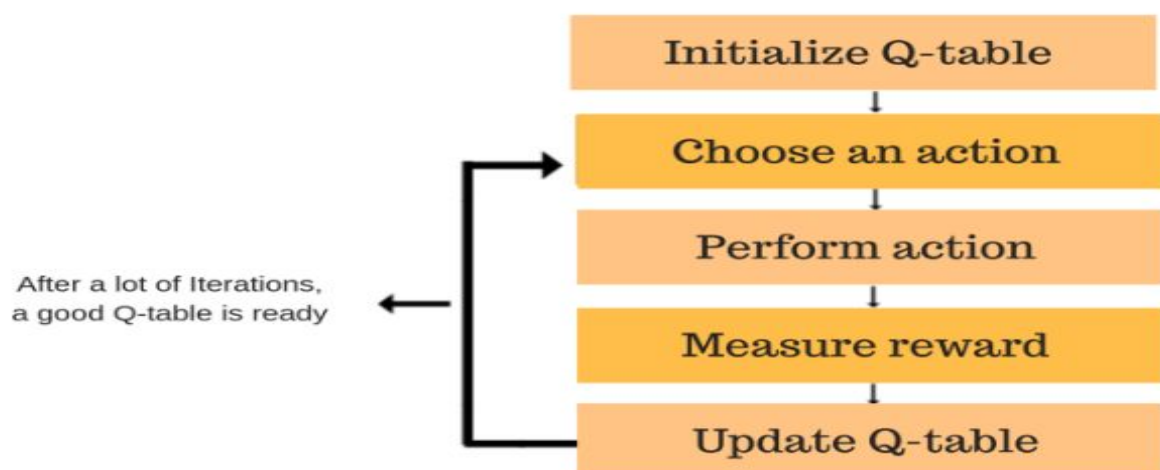


Figure 14

Initially (see Figure 14), a Q-table has to be created, the table could have its values in a form of a Markov Decision Process (MDP), hence the initial states are given a value based on the policy (Doya, 2007). Consequently, the agent chooses an action then performs the action - actions in this project are defined as the clockwise or counterclockwise motor rotation under different digital signal duty cycles (Shih-Yu Chang & Ching-Lung Chang, 2016). Followed by a reward measurement - in this project the reward is given based on the change in angle difference compared to

the old angle difference. Finally updates the Q-table (see Figure 15) (Shih-Yu Chang & Ching-Lung Chang, 2016). Thereupon, the action value is then used to determine the long-term effectiveness of the policy.

$$Q(s_t, a_t) = (1 - \alpha)Q(s_t, a_t) + \alpha[R_t + \gamma \max_{a'} Q(s_{t+1}, a_{t+1}')]]$$

Figure 15

The updating method uses the maximum of the following action value to update the previous value function using the following formula in Figure 14 above. Where $Q(s, a)$ represents the state and action pair at time t . After entering the state $s(t+1)$ from $s(t)$, a reward $R(t)$ is provided and the corresponding action $a(t+1)$ is performed according to a particular policy. Since the Q-Learning updating method uses the maximum action value, this method significantly reduces the time to achieve convergence which is why Q-Learning was used for this project (Doya, 2007; Shih-Yu Chang & Ching-Lung Chang, 2016). Notably, the loop repeats multiple times and becomes more accurate the more it iterates, which displays how when an agent keeps interacting with an environment it keeps learning and improving its policy.

2.4.3 Markov decision process (MDP)

Markov decision process widely used in processes which use Reinforcement Learning because MDP is a discrete-time stochastic control process (Feinberg, 1996); it deals with the existence of uncertainty either in observations that drive the evolution of the system. Moreover, provides a mathematical framework for modelling decision making in situations where outcomes are partly random and partly under the control of a decision-maker.

Chapter 3

3.0 Hardware

In this chapter is discussed what hardware was used, the reason behind choosing each of the hardware parts, modifications made, and setup of the hardware.

Since the purpose of the project is to implement PID controller and Reinforcement Learning of which the latter is high level based, Arduino was not that suitable for the purpose since the Arduinos as microcontrollers, have low memory and are not a good option when it comes to large based programs.

When it comes to Machine Learning & Artificial Intelligence and huge based programs, Raspberry Pi is preferred as it is a single-board computer with a 64 bit based ARM processor that can run Linux based operating systems. Thus, since Raspberry Pi can run Linux, it can support various operating systems with inbuilt libraries for Python, C, C+, etc. Whereas Arduino is a series of 8-bit microcontroller based boards that have no OS and use an IDE for programming due to the reason that it has low memory and large based programs cannot run.

Both Raspberry Pi and Arduino have good community support. However, Raspberry Pi has particularly excellent support for machine learning with vast community support available. You can easily learn from hundreds of online articles of Machine Learning on Raspberry Pi online. Additionally, Raspberry Pi uses an SD card which could have a memory range from 4 GB to 64+ while an Arduino has inbuilt flash memory in size of KB's.

The hardware that was used to control the robot:

1. Raspberry Pi 4 Model B
2. Gyroscope MPU-6050
3. Servo DXW90 (plastic) & Servo MG90S (metal) - both modified in order to be able to rotate with 360 degrees
4. Jumper cables
5. 6 AA 2900mAh batteries
6. Battery holder
7. UBEC 5V switch regulator

3.1 Setup

All the parts had to be set up appropriately in order for the robot to be able to read data from the sensors, analyse real time data, and move correspondingly based on the analysed data.

3.1.1 Raspberry Pi

As seen in Fig. 16, the Raspberry Pi has USB-C, two micro HDMI ports, two USB 2, two USB 3, Gigabit Ethernet, 40 pins, and SD card port at the bottom of the board. Initially, almost all of them were used in order to install Raspbian Linux. The USB-C was used as a power supply, a monitor was connected to either one of the micro HDMI

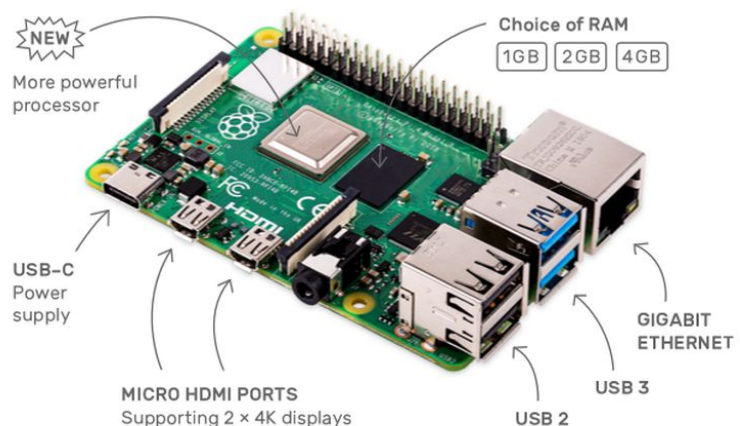


Figure 16

ports, a mouse and a keyboard was connected to either one of the USB 2 and USB 3 ports, and finally an ethernet cable was put into the Gigabit Ethernet port. Later on, all of the cables were removed except the USB-C power supply since the Pi was controlled remotely using Secure Shell (SSH).

Since for the initial prototypes only one motor, MPU-6050 sensor, and a USB-C were connected to the Pi, a breadboard was not needed even though it was prepared just in case.

Each of the physical pins had to be connected correspondingly so the sensors and the motors to work (see Fig. 17). The pins that were used are 3, 4, 5, 6, 12, 14 and in the next sections is discussed what was connected to those specific physical pins.

		Physical Pins			
GPIO#	2nd func	pin#	pin#	2nd func	GPIO#
N/A	+3V3	1	2	+5V	N/A
GPIO2	SDA1 (I2C)	3	4	+5V	N/A
GPIO3	SCL1 (I2C)	5	6	GND	N/A
GPIO4	GCLK	7	8	TXD0 (UART)	GPIO14
N/A	GND	9	10	RXD0 (UART)	GPIO15
GPIO17	GEN0	11	12	GEN1	GPIO18
GPIO27	GEN2	13	14	GND	N/A
GPIO22	GEN3	15	16	GEN4	GPIO23
N/A	+3V3	17	18	GEN5	GPIO24
GPIO10	MOSI (SPI)	19	20	GND	N/A
GPIO9	MISO (SPI)	21	22	GEN6	GPIO25
GPIO11	SCLK (SPI)	23	24	CE0_N (SPI)	GPIO8
N/A	GND	25	26	CE1_N (SPI)	GPIO7
EEPROM	ID_SD	27	28	ID_SC	EEPROM
GPIO5	N/A	29	30	GND	N/A
GPIO6	N/A	31	32	-	GPIO12
GPIO13	N/A	33	34	GND	N/A
GPIO19	N/A	35	36	N/A	GPIO16
GPIO26	N/A	37	38	N/A	GPIO17
N/A	GND	39	40	N/A	GPIO21

Figure 17

3.1.2 Motor

The servos were chosen because they are easier to control compared to the other types of motors. A Servo is a small device that incorporates a two-wire DC motor, a gear train, a potentiometer, an integrated circuit, and an output shaft. Of the three wires that stick out from the motor casing, one is for power, one is for ground, and one is a control input line. As seen in Fig. 17 the motor has been connected to pins 2 (5V power), 12 (GPIO), and 14 (GND) (see Fig.18).

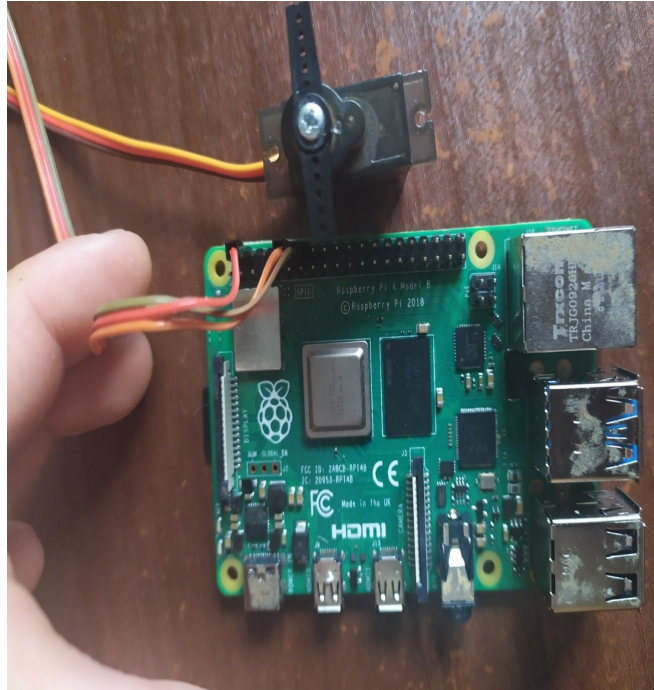


Figure 18

Pin 12 is a General-purpose input/output (GPIO) which can be designated as input or output and used in a wide range of purposes. In the case of this project, pin 12 was designated to send signals to the motor so it can move correspondingly.

Pin 14 - Ground (GND) is used as a safety measure to prevent accidental electrocution. If for some reason the body becomes electrically charged, there would be nowhere for the electricity to go until someone touches it and is shocked. To prevent this, a wire is used to connect the chassis to the ground, thus if by any chance there is any rogue electrical charge, it would dissipate to the ground (GND).

3.1.3 Sensor

The MPU-6050 (see Fig. 19) has six-axis and contains two sensors - Gyroscope and Accelerometer, both of which were used in order to read the data to the highest precision. The sensor helps us to measure velocity, orientation, acceleration, displacement, and other motion like features.



Figure 19

MPU-6050 has been connected to 4 pins - 3, 4, 5, 6 (see Fig. 20). Importantly, in order for the MPU-6050 to be connected to the PI, it had to be soldered first.

Pin 3 is directly connected to the SDA (serial data) which is the line for the master and slave to send/receive data.

Pin 4 provides 5V and is connected to the VCC (Voltage Common Collector) of the MPU-6050 sensor, thus the sensor is powered on.

Pin 5 is connected with the SCL (serial clock) which provides serial communication.

Both Pin 3 and Pin 5 are I2C (Inter-Integrated Circuit) based. I2C is a serial protocol for a two-wire interface to connect low-speed devices.

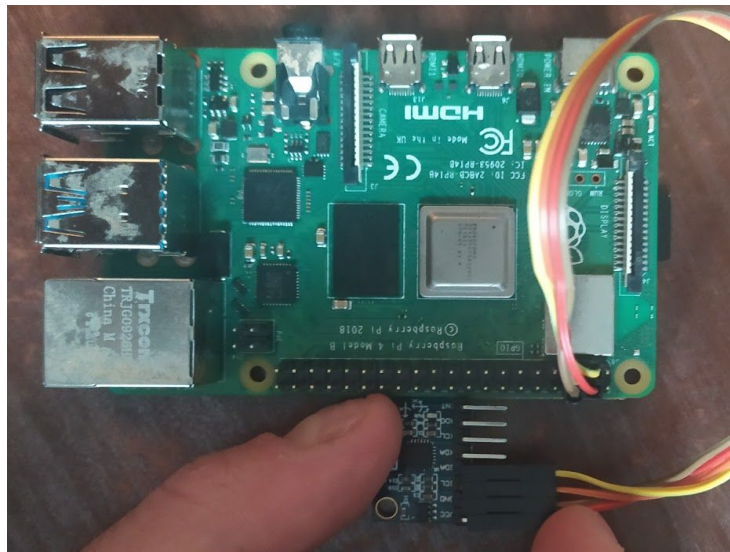


Figure 20

Pin 6 is Ground (GND) and is connected to the GND of the MPU-6050 sensor, as a safety measure to prevent accidental electrocution.

Notably, the sensor data initially is raw and requires a filter for the data being read since there is always noise which could cause imprecise data reading which is an issue and addressed later on in this report.

3.1.4 Batteries

The voltage required for the PI to work effortlessly is 5V anything above that voltage risks burning the PI. Since the battery holder used (see Fig. 21) contains 6 slots each for 1 2900mAh battery (1.2V, the total voltage passed on would be 7.2V. Therefore, a UBEC 5V switch regulator (see Fig. 22) was required to balance that voltage. The switch regulator then is connected to the PI using a micro USB port which is directly connected to the power supply of the PI.



Figure 21

Whenever the batteries ran out of power, a battery charger (see Fig. 23) was used which could charge up to 4 batteries at a time. The charger is connected to, for instance, a PC, using a USB.



Figure 22



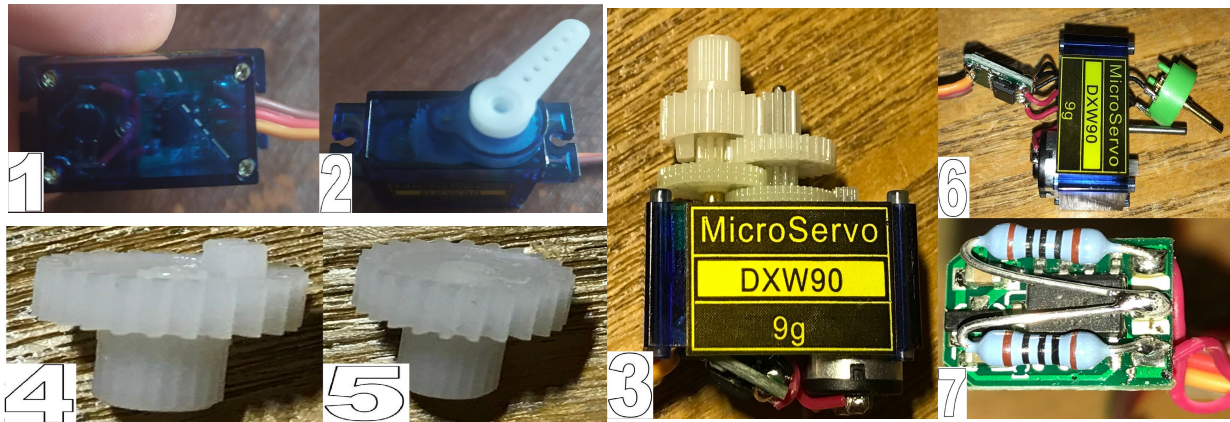
Figure 23

3.2 Modifications

Since both servos used in the development cannot move 360 degrees modifications had to be made. The only difference between the two servos is that one of them is made out of plastic and the other ones is made out of metal, the implemented modifications are the same for both of them.

1. Remove Screws and then remove Top And Bottom (Figure 24: 1-3)
2. Remove Coggs.
3. Remove Stop on Cogg (Figure 24: 4-5)
4. Replace Coggs
5. Pop Out the Potentiometer - A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider(Figure 6).
6. The two resistors then were soldered to replace the potentiometer for the continuous rotation (Figure 24: 7). The “neutral point” of the servo will depend on the ratio of the 2 resistors as they act as a fixed voltage divider.
7. The Potentiometer may or may not be cut off from the board. The key is to disconnect it from the circuit and replace
8. it with the fixed voltage divider made from the 2 resistors. In some servos you may need to leave the potentiometer in place for mechanical support reasons.
9. All other parts are then put back together.

Figure 24



3.3 Issues

For the initial prototype, a monitor, a mouse, an ethernet cable, and a keyboard were connected to the Pi which was adding additional side support for the robot. Hence, they had to be removed. Once removed the robot was intended to be controlled using SSH which requires internet connection. However, there was an issue with the Pi since it could not find the local Wi-Fi spots. Thus, the issue had to be fixed manually and make the Pi connect to the specific Wi-Fi spot automatically when powered on. In order to do so several steps had to be taken.

Initially, the Raspberry Pi had to be started and connected to a display and a keyboard. Using the terminal, the network interfaces file had to be edited. By typing the command:

```
$ sudo nano /etc/network/interfaces
```

 Figure 24-1

Will open a file that contains all known network interfaces, in which the first line had to be changed to:

```
auto wlan0
```

 Figure 24-2

Afterward, at the bottom of the same file, more lines had to be added (see below) in order to tell the Raspberry Pi to allow wlan as a network connection method and use the /etc/wpa_supplicant/wpa_supplicant.conf as a configuration file.

```
allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp
```

Figure 24-3

After adding the lines above, a configuration file had to be created, namely, the `wpa_supplicant.conf` file. By typing

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Figure 24-4

A new file will be created in which specific parameters had to be added:

```
network={
ssid="YOUR_NETWORK_NAME"
psk="YOUR_NETWORK_PASSWORD"
proto=RSN
key_mgmt=WPA-PSK
pairwise=CCMP
auth_alg=OPEN
}
```

Figure 24-5

Finally, considering that all the instructions in the files above were saved successfully, by rebooting the Raspberry Pi was able to connect automatically to the wireless network.

Additionally, in order to connect using SSH, the Raspberry Pi connection options had to be edited from its menu so it allows SSH connection.

Chapter 4

4.0 Design

In this chapter are discussed the goals of the design, the designs of the robot which were created using FreeCAD and then 3D printed and other relevant information.

The goal of the design was to make the robot's aesthetics be as close as the Robert Gordon robot's aesthetics, also referred to as "Glitch" (see Fig. 25). The robot is supposed to have one wheel situated between the two legs, which are connected to the body of the robot where the Raspberry Pi, batteries and sensor would be situated. The arms, neck, and head are just decorative.

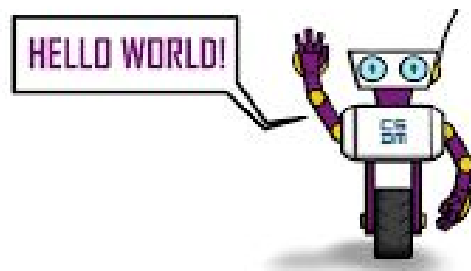


Figure 25

Since the shape of the robot has to resemble "Glitch", the motors had to be inserted into the bottom part of the leg and connected directly with the wheel's rod, which goes directly through the middle of the wheel. Thus, whenever the motor moves, it also moves the rod backward or forward which moves the wheel correspondingly.

Two different techniques were used to connect the motor with the rod, initially, there was a hole with a specific size so the motor can attach to it, the glue had to be added as well, and the second and better technique was to just attach the motor to the rod using screws. The latter allowed more flexibility since if there was an issue, the motor could be just detached from the motor holder.

The legs of the robot in all designs had particular holders so they are well placed and immobile. The initial design had two holes inside the bottom part of the belly of the robot which were placed with a specific distance between each other. Whereas in the final design the legs holders were extensions of the robot's belly and the legs had to be slid properly onto the holders.

The legs, body, and the decorative parts - arms, neck, head, were all made hollow which was compulsory since the jumper cables had to go through them instead of having the cables hanging externally. The wheel of the robot had to be as thick in size as possible because thickness helps the robot maintain balance, whereas the diameter of the robot depends mostly on the motor's torque - the larger the wheel

in diameter the more torque is required to move it. With each new design the size of the wheel had to be decreased so it is easier for the motor to move it.

Each of the robot's parts had to be designed separately and while doing so it had to be taken into consideration that each of the parts had to be able to connect with the parts to which it is supposed to be connected. Importantly, multiple lessons were learnt during this process, for example, when designing parts that are going to be 3D printed, a part should be able to lay on the ground without any support otherwise the printer would print the support per se. Which would result in using more material and the hassle of removing the support afterward which sometimes could be quite hard and could result in damaging the part itself.

In addition, 3D printing the parts required multiple iterations because there were inaccuracies which were not spotted when designing the parts.

4.1 First Prototype

The initial prototype was with an approximate height of 35 cm (Fig. 26). The wheel's radius was 8 cm and the height was 5 cm and the rod's radius was 2.4 cm and a height of 2 cm. The rod contained a small hole so the servo can be attached to it. One of the lessons I learnt in designing the parts is that, for example, the hole in the rod had to be made 0.2 cm greater in size than the size of the servo part that connects to it because when printing there should always be left roughly 0.2 cm extra space due to the reason that the plastic shrinks slightly after it is printed.

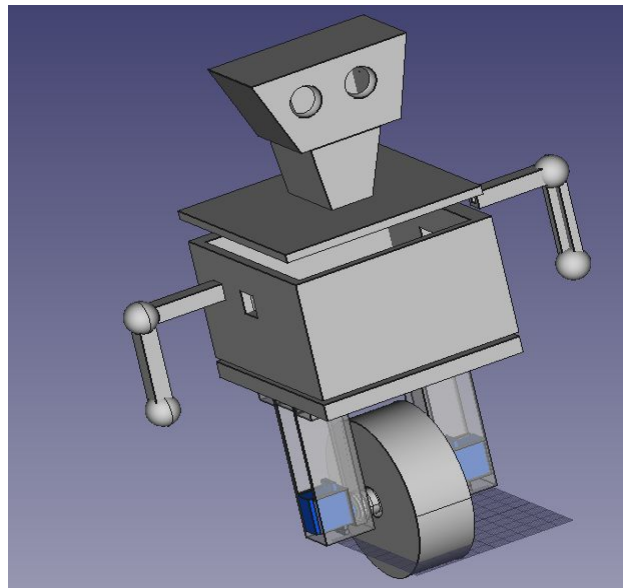


Figure 26 (Servos in blue color)

The legs are 8.5 cm high, with a 2.5 cm length, and with a 1.8 cm width. On their top they have a smaller extension that goes into the belly of the robot, the extension is inserted through the belly leg holes (see Fig. 27). Once the legs were placed properly, glue was used to make them immobile.

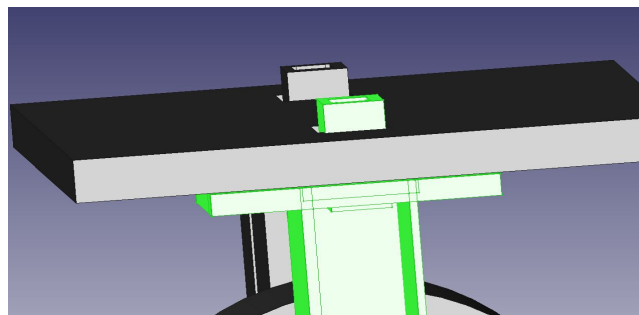


Figure 27

The most difficult part of the design was to make a proper holder for the servo motor at the bottom of the leg (Fig. 28). The servo motor goes into the specially designed holder which has a small hole at the bottom back part for the cables that go into the servo, side support so the servo does not move left or right hitting the sides of the leg, and a large hole in the middle bottom part of leg, for the side plastic of the servo (Fig. 29).

There is enough space inside the leg for the rod to lay upon, thus the motor did not hold the entire weight of the robot.

Notably, even though the servo was placed nicely into the holder, an issue was found when the motor started receiving signals to move which caused the servo to detach from the rod quite often. As a short-term fix glue was used on the sides of the servo which temporarily fixed the issue, however, it made it extremely difficult to remove the servo when needed.

The belly of the robot on which are placed the Raspberry Pi, batteries, and sensor, is 12 cm in length, 12 cm in width, and is 1 cm in height. As it has two holes on the for the legs situated in the middle part (Fig. 30)

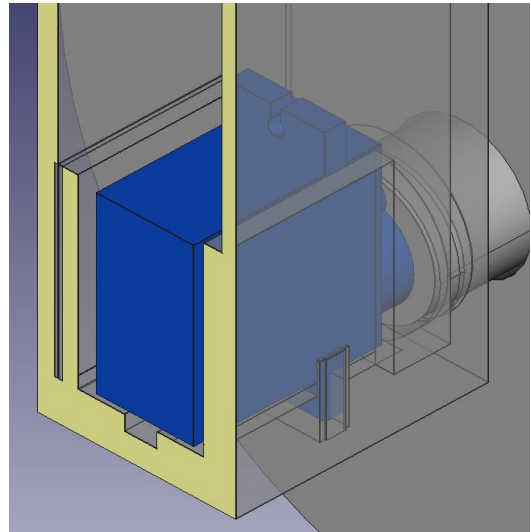


Figure 28

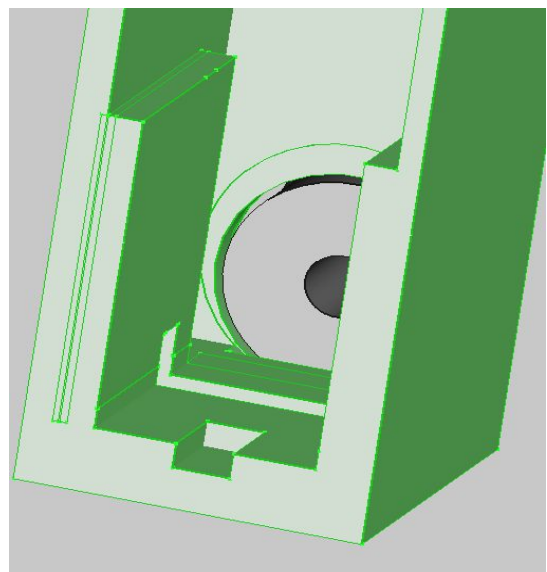


Figure 29

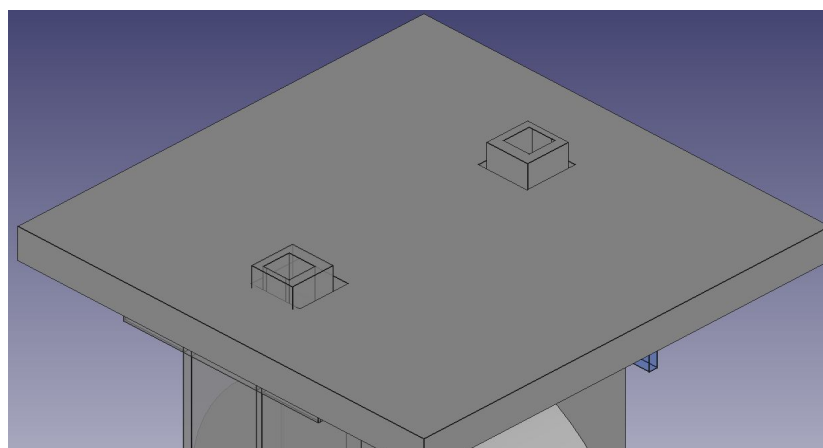


Figure 30

Due to the fact the arms, neck, head were only decorative, the plan was to print them once everything else works and then paint the robot with the colours of the Robert Gordon robot “Glitch” (purple, yellow, blue, black). For testing purposes only the wheel, rod, legs, and the top on which were placed the Raspberry Pi, batteries and sensor. All the changes that had to be made eventually were made with that specific lower body of the robot.

4.2 Second Prototype

The more sophisticated second design prototype fixed many of the issues the initial first prototype had (Fig. 31). The total height of the design is approximately 17 cm. Since the first prototype had an issue with friction and rubber bands had to be used, in the second design, the wheel was separated into two parts, namely, inner and outer tyres. The inner tyre’s radius 5.32 cm and had a height of 4 cm. Whereas the outer tyre’s radius is 5.92 cm and the same height as the inner. However the outer tyre was made with a softer plastic so it can be slightly more flexible, thus easier to insert on the inner tyre.

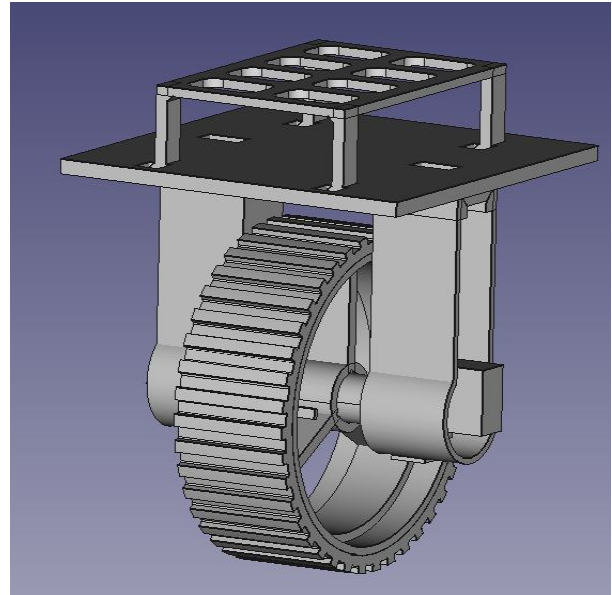


Figure 31

Additionally, to increase the friction, the outer tyre was made edgier as seen in figure 31. Initially, the wheel was way smaller, however, it turned out that even though there is way less torque required for the motor to spin it, the smallness was not that effective. The surface area the wheel was occupying was not enough and it had to be increased in order to improve the balance.

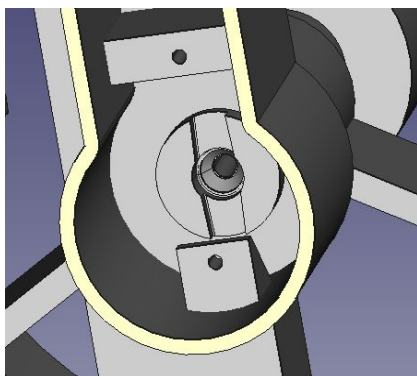


Figure 32

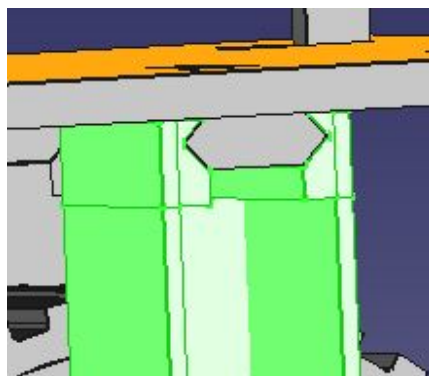


Figure 33

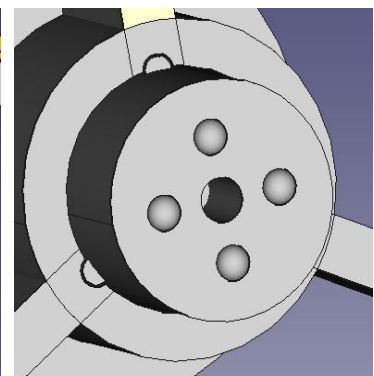


Figure 34

As mentioned in the previous section, the motor holder had a design flaw as the motor kept getting detached when powered on and moving. A better motor holder had to be designed as seen in Fig. 32. The motor is put and screwed using the two holes on the top and the bottom. A propeller is inserted on the motor and goes into the vertically curved space inside the disc. The disc contains 4 buds on the outer side that go into the rod buds space (Fig. 34). Hence, when the motor is powered on and receives a signal to move, it spins the propeller which spins the disc which spins the rod which is part of the wheel, thus the wheel moves correspondingly. Since just screws were used to attach the motor to the motor holder that gave the flexibility to easily remove the motor if necessary. The initial prototype 2 can be seen printed in Figures 35-37.

The legs had a holder to which they had to slide in (Fig. 33). Which yet again, provided stability and flexibility compared to prototype 1 in which they had to be glued. The flexibility helped as it made it quite easy to change the legs because multiple times later on longer legs had to be printed.

At the very top of the belly of the robot can be seen as a plastic grid on which was inserted the battery holder filled with the 6 batteries. The battery holder which was using double-sided duck tape in order to not fall if the robot oscillates. Below the plastic grid was placed the Raspberry Pi and on the side was placed the sensor, both of which were using double-sided duck tape to be properly attached to the body of the robot. The grid was placed into special holes that were designed so the grid is immobile when the robot moves.

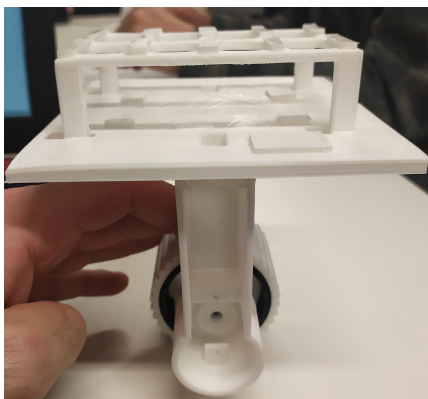


Figure 35

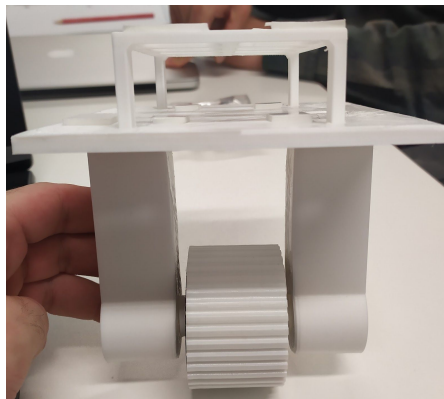


Figure 36

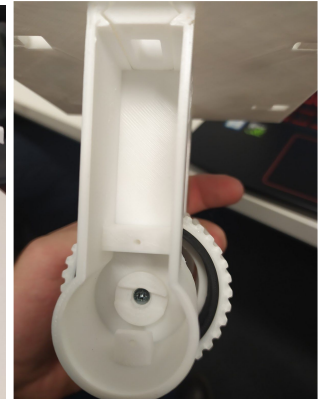


Figure 37

4.3 Issues

The first prototype had a large number of flaws that decreased the robot's ability to maintain self-balance.

First of all, using glue to attach the legs to the body of the robot was quite an inflexible idea as when it comes to changing parts force had to be used required to separate the parts. The motor kept detaching during testing as the support for it to stay still was not good enough. Glueing the motor to the rod caused a major issue since initially a plastic motor was used which was wearing out and could not be taken out.

Secondly, the glue had to be added to the sides of the motor as it was hitting the sides of the leg when moving, after additional support was added the motor became more stable but still continued to move and hit the walls of the already added leg wall support. Eventually, glue was used on both sides of the motor in order to stop the movement which could damage the servo in the long term.

Thirdly, the cables that were coming out of the motor did not have additional space on the bottom of the leg, thus this created a flaw which was eventually fixed, however, the cables had marks and if the additional space was not added after a longer period of time the cables would have torn apart.

Moreover, since the rod is printed separately from the wheel and glued together with the wheel, in one of the printing iterations the rod was 0.004 cm off the center of the wheel which caused issues with the robot's balance.

Initially, friction of the wheel was not enough, therefore rubber bands were put around it to fix the issue. To fix that issue in the long term in the second prototype was used, as mentioned above, an outer tyre with edges that were made out of softer plastic and was put on top of the inner tyre.

In the second prototype, two issues occurred, one of them was the length of the legs, which had to be extended twice from 9 cm to 21 cm and then again to 30 cm since the longer object has a greater moment of inertia. Therefore, its rate of angular acceleration will be less for the same turning force which gives you more time to make adjustments to maintain the state of balance.

The second issue was that the small piece that is connected to the rod of the wheel kept getting out of the 4 buds it was connected to due to the reason that the 4 buds in the rod were too shallow, yet again, as a temporary fix was used glue and afterward was printed a better-suited part that did not get out when in motion.

Multiple iterations had to be completed until the different design flaws were found. From each iteration I learnt new lessons about designing a robot from scratch, the main lesson is that designing robots from scratch requires extreme attention to detail and persistence, one has to think about every single detail and make sure that it will be compatible with the rest taking into consideration physics, hardware, mechanics.

Chapter 5

5.0 Implementation

In this chapter is discussed how the project was implemented using specific tools, such as FreeCAD, Python, C# and Unity, and how the development practices required to be implemented in order to make the robot self-balancing.

5.1 FreeCAD

FreeCAD is an excellent free tool to make 3D objects, the extension of the file generated is .FCSD which is then used in order to print the 3D model. As a person with no personal experience in 3D modeling, there were many techniques that had to be learnt and many features that had to get familiar with. Due to the reason that the models were going to be printed, each of the models had to be designed so it is able to lay on the ground without support, taking into consideration that if an object's shape is cylinder based, such as the robot's wheel, it has to lay on the ground horizontally. The 3D printers work on layers, adding a layer on top of another layer, therefore if the object has a cylinder shape and is standing vertically, the object would be printed and not have a complete, smooth cylinder shape.

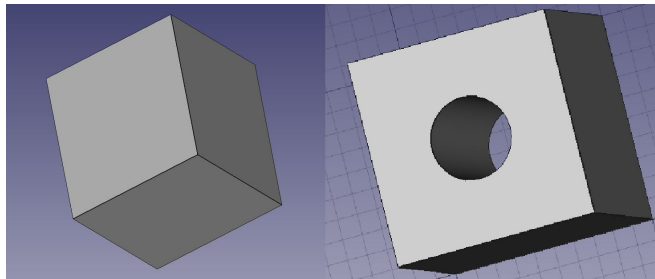


Figure 38

For the first prototype, one of the parts that was considered to be used as an addition to the wheel rod can be seen in Figure 38. Eventually, the part was not used as it was adding additional design complexity which was unnecessary. However, it is a good example of how using several basic geometry shapes can create unlimited amounts of other shapes using FreeCAD. For instance, in order to create the part in Figure 38, a cube and a cylinder were created, the cylinder was then positioned inside the cube and by using FreeCAD's feature "Cut", a brand new shape was created (Fig. 39).



Figure 39

Any of the basic default objects, such as cube, cone, cylinder, etc, can have their details changed correspondingly with ease, based on the development goals (see Figure 40-41 as an example).

Property	Value
Base	
Angular Def...	28.65 °
Bounding B...	false
Deviation	0.20
Display Mode	Flat Lines
Draw Style	Solid
Lighting	Two side
Line Color	■ [25, 25, 25]
Line Width	2.00
Point Color	■ [25, 25, 25]
Point Size	2.00
Selectable	true
Selection St...	Shape
Shape Color	■ [204, 204, 204]
Transparency	0
Visibility	true

Figure 40

Property	Value
Attachment	
Map Mode	Deactivated
Base	
▼ Placement	[(0.00 0.00 1.00); 0.00 °; (0.00 mm 0.00 m...
Angle	0.00 °
▼ Axis	[0.00 0.00 1.00]
x	0.00
y	0.00
z	1.00
▼ Position	[0.00 mm 0.00 mm 0.00 mm]
x	0.00 mm
y	0.00 mm
z	0.00 mm
Label	Cube
Box	
Length	10.00 mm
Width	10.00 mm
Height	10.00 mm

Figure 41

Any of the details of each particular object could have been changed, whereas the most used ones for this project were:

- Transparency (Fig. 40, bottom part) - used to make some of the parts more transparent in order to check the compatibility of the part per se with the other parts surrounding it.
- Placement (Fig. 41, middle part) - used to adjust the axis, angle, and position of the object depending on what is needed.
- Length, Width, Height (Fig. 41, bottom part) - used to adjust the length, width, height of the object, depending on the type of the object per se. Moreover, depending on what the object is, it could have an additional field such as radius.

FreeCAD has multiple other features that were used in the development of this project which enabled me to successfully create more complex objects effectively and in a faster manner. Some other features that were used are:

- Mirror object - by completing one of the legs of the robot, “mirror object” allowed it to quickly mirror to leg and have an absolute duplicate in seconds, all it had to do afterward was to change the axis so the newly created leg is 180 degrees rotated.

- Merge - by selecting two or more objects they could have easily been merged into a new one by just using the merge feature. When creating the extension for the legs so they can be easily attached to the body of the robot the “Merge” feature helped tremendously.

To summarise, FreeCAD was tremendously useful software for the development of this project which allowed me to design the parts for the robot fast and efficiently. It allows each part to be highly customizable depending on what is needed. Working with such an efficient tool taught me a lot about design modeling.

5.2 Python

Python was selected for this project because as an interpreted, high-level, general-purpose programming language it has the capabilities to quickly develop a project and also because it provides a lot of development libraries that are easy and fast to use with a lot of documentation online. Languages such as C++ and Java were considered as well, however, python’s features outweighed the aforementioned.

Before the PID code is used, the data from both the gyroscope and the accelerometer goes through a complementary filter which removes the noise. Once the noise is removed, the current state of the robot is determined based on already pretested data. The pretested data are the specific gyroscope and accelerometer data which stands for the goal state of the robot. When the robot is manually held vertically as it should be when maintaining self-balance, the sensor is standing still taped to the robot in a particular position and returns the current accelerometer and gyroscope data. Then the data of both of them had to be used to find the goal state of the robot. Using that specific goal state and the current state when the robot is powered on, an error could be assigned and used in both the PID controller and Reinforcement Learning.

5.2.1 PID implementation

As discussed in Chapter 2, the PID controller is a widely used controller in a closed-loop control mechanism based systems. Each of its three components (P - proportional, I - integral, D - derivative) has to be adjusted manually for each system. In the case of this project as there is a goal state which is the robot standing vertically and maintaining balance, and current robot state based on the sensor data, an error is generated based on their difference. The “P” tries to fix the error, the “I” helps “P” to not overshoot and “D” helps by decreasing the process value if it is increasing too quickly, thus making the PID output value more steady.

As of the PID implemented in the case of this project (Fig. 42), as of line 23, the error which equals the goal state minus the current state, and the “dt” current date-time (Iteration time) is added to the integral value. Then on line 24 the derivative is assigned to the result of the current error minus the prior error divided by the “dt” iteration time. The error prior is then assigned to the current error so it can be used in the next iteration. Finally, the PID value is calculated using the aforementioned variables and values and then is returned processed further in order to send particular signals to the motor.

```

22:     def update(self, error, dt):
23:         self.integral += error * dt
24:         derivative = (error - self.error_prior) / dt
25:         self.error_prior = error
26:         self.pid_value = self.Kp * error + self.Ki * self.integral + self.Kd * derivative
27:         print("Time = " + str(dt) +
28:               "\nIntegral = " + str(self.integral) +
29:               "\nError Prior = " + str(self.error_prior) +
30:               "\nValue = " + str(self.pid_value))
31:         self.update_time_frame()
32:         return self.pid_value

```

Figure 42

The value from the PID Controller is then mapped so the number is compatible with the PWM signal. Initially, a python library was used which uses “Duty Cycles” in order to send those particular PWM signals to the motor.

A duty cycle (see Fig. 43), also referred to as a power cycle, is a fraction of a single period in which a signal is in an active state. It can be expressed as a percentage or a ratio, in this case the library which is being used expresses the duty cycle ratio with a ratio from 2.5 to 12.5. A period is the time required for a signal to complete an on-and-off cycle. In electronics, the duty cycle is a percentage of the ratio of pulse duration, or pulse width (PW) to the total period (T) of the waveform.

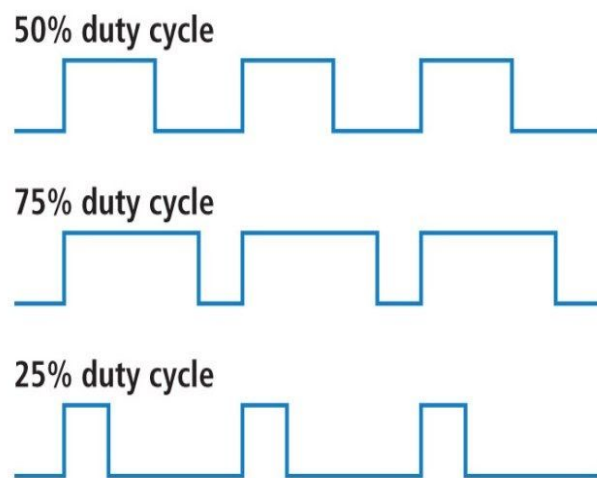


Figure 43

Eventually, in the final stages of the implementation, a different library was used which is even more low level based than the previously used Duty Cycle library. In other words, the switch was from software PWM to hardware PWM. The reason for

switching between the two is because an issue with the motor responsiveness was found and the new library improved the speed of the signal.

5.2.2 Reinforcement Learning

The implementation of Reinforcement Learning was in the initial stages and did not continue due to design flaws and the responsiveness of the motor which was not fast enough. However, the code for the key concept was partly written.

In order to implement Reinforcement Learning, Q-Learning was used, which as discussed in chapter 2 is an off-policy reinforcement learning algorithm that seeks to find the best possible action to take given the current state.

To implement Q-Learning, a Q-matrix was created which follows the shape of [state, action] in which all the values are initialised to be equal to zero. After each episode the values in the Q-matrix get updated as the matrix is used as a reference for the robot to select the based action based on the q-value.

The next implemented step is making updates to the Q-matrix. When the robot interacts with the environment, i.e. tries to self-balance, updates are made to the state-action pairs in the q-matrix. The robot selects the action based on the max value of those actions. Hence, there are only three actions for any given state - move forward, stay still, move backward. Once the robot starts interacting with the environment, updates occur after each action and end when the robot reaches a terminal point such as falling down. Multiple episodes have to be implemented in order for the robot to converge and learn the optimal Q-values. However, due to the design flaws and the responsiveness of the motor, those episodes were not implemented in order to get the optimal q-value data.

A basic update of the Q-matrix is the can be seen in Figure 44, the code from the figure is found in the software simulation written in C#.

```
this.Quality[currentState][nextState] = ((1 - this.LearningRate) * this.Quality[currentState][nextState]) //current_q
+ (this.LearningRate * (this.RewardMatrix[currentState][nextState] + (Gamma * maxQuality)));
```

Figure 44

Several variables above were not mentioned. The learning rate means how much the new value is accepted versus the old value. The “Gamma” is a discount factor and it is used in order to balance immediate and future rewards. The “RewardMatrix” consists of values that are received based on a certain action at a given state. The “maxQuality” is used to multiply the value from the reward matrix in order to get the maximum future reward.

5.2.3 Filters

Complementary filter was implemented in order to remove the noise from the Gyroscope and Accelerometer (Fig. 45). The complementary filter recognizes the dominant rate. Moreover, average input and duty cycle filters were implemented as well to increase the accuracy of the data being read.

The complementary filter takes slow-moving signals from the accelerometer and fast-moving signals from a gyroscope and combines them in order to recognise the dominant rate. The accelerometer gives a good indicator of orientation in static conditions. Whereas, the gyroscope provides an indicator of tilt in dynamic conditions. Therefore, to pass the accelerometer signals through a low-pass filter and the gyroscope signals through a high-pass filter then combine them to give the final rate.

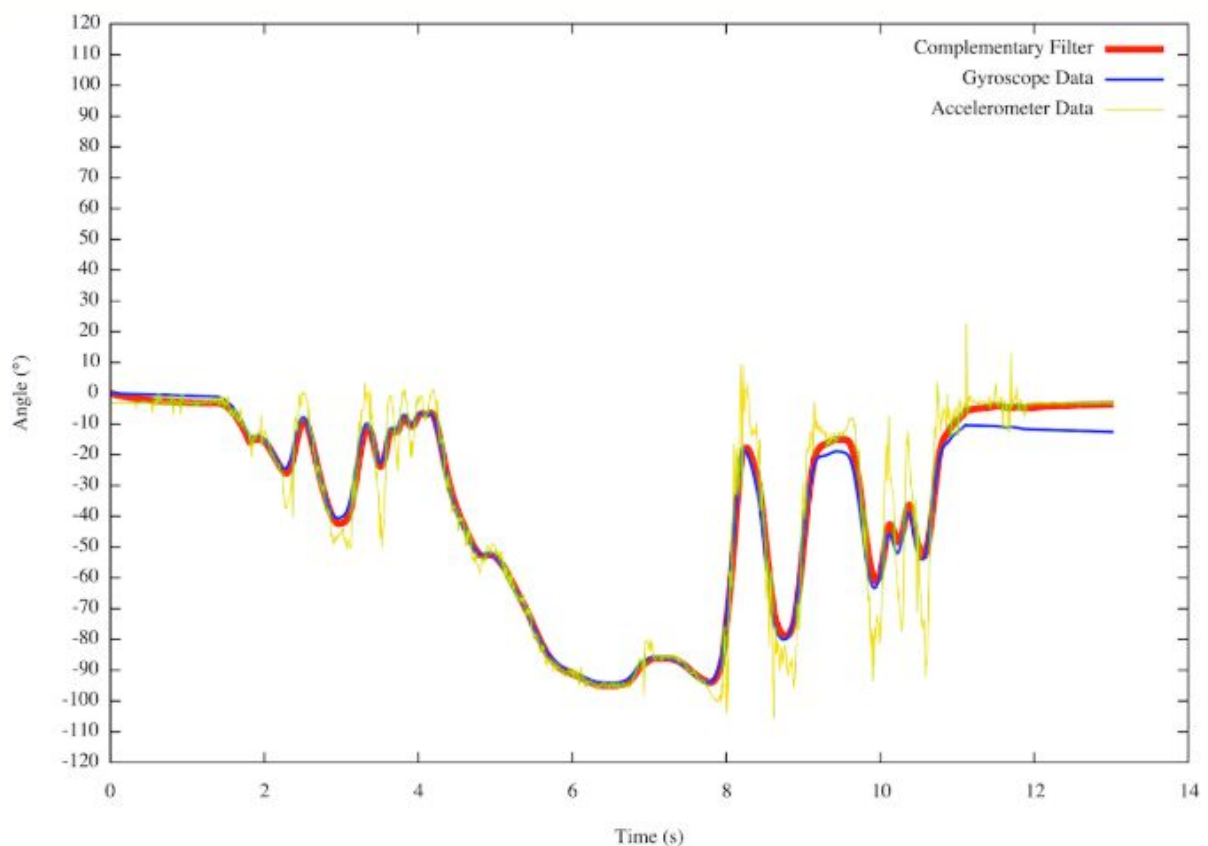


Figure 45

As the frequency response of the low-pass and high-pass filters adds up to 1 at all frequencies which means that at any given time the complete signal is subject to either low pass or high pass.

The Complementary filter uses the best of both gyroscope and accelerometer. The data from the gyroscope is used because of its precision and not susceptible to

external forces. Whereas in the long term, the data from the accelerometer is used, as it does not drift. The filter implemented looks as follows:

$$angle = 0.98 * (angle + gyrData * dt) + 0.02 * (accData)$$

Figure 46

The gyroscope data is combined with every timestep with the current angle value. Afterward, the data is combined with the low-pass data from the accelerometer which has already been processed with atan2 (angle in the Euclidean plane, given in radians). The constants used in Fig. 46 (0.98 and 0.02) have to add up to 1, if required they can be changed to tune the filter properly, however in the case of this project 0.98 and 0.02 were appropriate constants.

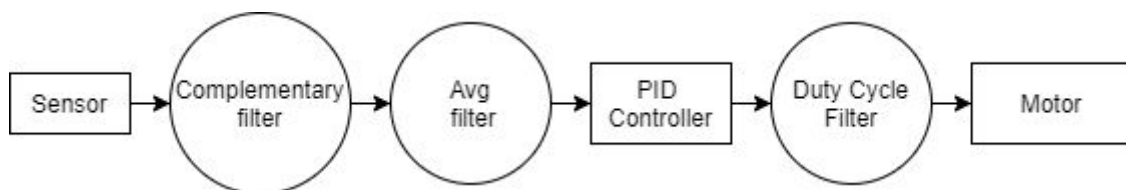
The whole complementary filter function has to be used in an infinite loop, thus with each new iteration the pitch and roll angle values will get updated with the new gyroscope values by means of integration over time.

The complementary filter then inspects the magnitude of the force seen by the accelerometer per se and whether it has a reasonable value that could be the real g-force vector. If the value is greater or smaller that is an indicator for a disturbance that has to be taken into account.

Afterward, it updates the pitch and roll angles with the accelerometer data by taking 98% of the current value, and adding 2% of the angle calculated by the accelerometer, depending on the constant values that were used. This will ensure that the measurement is not going to drift, however, it is going to be very accurate in the short term.

Additionally, an average input filter and duty cycle filter were added in order to increase the accuracy and make the signals sent to the motor more smooth. The average input filter takes the last 5 sensor readings after they went through the complementary filter (Fig. 47). Whereas, the duty cycle filter takes the average of the last five duty cycle signals sent to the motor. As default values for both of them were set to the goal state values.

Figure 47



5.3 C# and Unity

The software simulation was implemented in Unity which has excellent support, documentation, libraries, assets, etc, and could easily be used with C#.

Initially, the robot had to be designed using different unity 3D objects and then put together (see Figure 48). Yet again, the design of the robot was designed to resemble the Robert Gordon robot “Glitch”. Apart from the design of the robot, a simple environment was created with which the robot can interact.

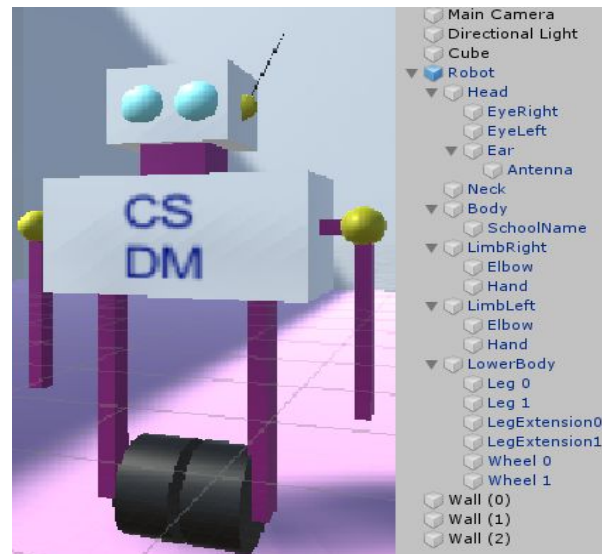


Figure 48

An essential part of the simulation was using realistic physics in order for the robot to effectively try to self-balance itself in an environment as close to reality as possible. Thus, rigidbody was used to enable the robot object to act under the control of real physics. The Rigidbody can receive forces and torque to make the object move in a realistic way. The robot object contains a Rigidbody to be influenced by gravity and act under added forces via scripting.

The scripts written for the simulation were PID Controller and RL (Reinforcement Learning). Similarly, the code in the PIDController corresponds to the code written in the hardware simulation except it is written in C#. As for the RL script, the code implemented is explained in more detail in the Reinforcement Learning section of this chapter.

Hence, since only the wheels of the robot are supposed to move, wheel colliders had to be added which are special colliders for grounded vehicles. They have built-in collision detection, wheel physics, and a slip-based tire friction model and can be used for objects other than wheels, but it is specifically designed for vehicles with wheels.

However, some other issues occurred with the physics which were causing the robot to fall instantaneously sometimes when it is in a self-balancing state and is tilted to either side. The bug was not found on time to implement both PID and RL.

Chapter 6

6.0 Testing and Results

In sections are discussed the results and the tests which had to be implemented. Most of the testing had been focused on the physical simulation as it required more PID tuning because of the different design changes which reflected on the robot's balance. Testing has also been done on the MPU-6050 and the software simulation.

6.1 Sensors

The goal state of the robot was of paramount importance as the PID required an error value using which to produce adequate results which is then used to send signals to the motor and thus balance the robot. Moreover, reinforcement Learning needed the goal state as well in order to aim to get to that particular state and receive more reward.

Multiple tests had been implemented by manually holding the robot in a vertical position and calculating the average number of multiple sensor results overtime. Three states were considered when testing, namely - vertical, forward, backward. The vertical state stands for the goal state (i.e. the robot is balanced), the forward state stands for the robot being at the boundary of not being able to regain balance when falling forward, and similarly the backward state stands for the point of no return when following backward. As the data from those states is crucial, especially the goal state, 15 tests were implemented for each of the 3 states for both the Gyroscope and Accelerometer and their average is calculated and used correspondingly.

Eventually, as the height of the robot was changed multiple times and the position of the sensor as well, therefore the tests had to be implemented again in order to make sure that goal state and the points of no return states have not changed or if they did change appropriate updates were made.

Additionally, since three filters were used to filter the data, increase the accuracy, and smoothen the signal sent to the motor, tests were implemented in which the third duty cycle filter was removed in order to test whether the speed of responsiveness would be more promising as well.

6.2 Physical Simulation

In order to map the PID value so the result is compatible with the Duty Cycle number, the PID value was divided by 20. The reason that a specific number was used is because, after careful analysis and testing of the current states of the robot, the number was ideal for making the PID value within the bounds of the duty cycle limits. Since the motor (DXW90) was changed once, the mapping value had to be tested again and changed appropriately, as for the second motor (MG90S) the mapping value was set to 15 as that specific number produced better signal results. Notably, the smaller the mapping value the higher the sensitivity of the signal sent to the motor.

Each of the two servo motors (Servo DXW90 & Servo MG90S) responded differently to the Duty Cycles signals which slowed the process of development down.

Typically, motors which use the same duty cycle library as the one used for this project, move based on duty cycle signals from 2.5 to 12.5, where 2.5 would be maximum speed backward, 12.5 would be maximum speed forward, and 7.5 would be a signal that would make the servo motor not moving. The values in between would either decrease or increase the speed based on how close the values are to the center (i.e. 7.5). The untypical in the case of the tests implemented was that both servos did not respond adequately to the values aforementioned. After careful testing numerous times was discovered that the servos respond to completely different duty cycle values with an extremely low range.

The DXW90 had a maximum forward signal at 7.6, maximum backward signal at 6.6 and no signal that would make the servo not moving. Hence a dead zone had to be added which was set to be from 7.08 to 7.12 in which the servo would not send any signals to the motor in order to stay still when at goal state.

The MG90S had a maximum forward signal at 7.3, maximum backward signal at 6.3 and stay still signal at 6.8 at which the motor did not move, thus marking the goal state.

Many issues were found along the way because of which no Reinforcement Learning (RL) tests were implemented. Hence, there also no RL results as well.

6.2.1 PID values

The tests for the first prototype indicated that the robot was able to maintain balance with a $(P) = 42$, $(I) = 18$, $(D) = 1$. A huge amount of values and combinations were tested in order to find the best PID tune, where the test values for each of the three parts of the PID were ranging from 0 to 300+. Importantly, when the tests were being implemented, the (P) was being set up first as it is the most important of the three as it tries to fix the error. Followed by the (I) as it is the second most important because it helps reduce overshooting and thus the robot oscillates less. Finally, the (D) whose effect on the PID value was quite insignificant.

The aforementioned PID values were approximate as when design changes were made and flaws were fixed, the values per se either decreased or increased slightly based on the changes made.

The tests with PID values for the second prototype indicated that the robot was able to stay straight the longest with slightly lower numbers, namely, $(P) = 29$, $(I) = 11$, $(D) = 0$.

As mentioned in the previous chapters, the legs of the robot had to be extended a several times due to the fact that the longer the object the greater moment of inertia, therefore the rate of angular acceleration is less for the same turning force, which in return gives more time to make adjustments to maintain the state of balance. As a result, the tests implemented with longer legs were more promising as the robot had more time to respond to the changes the longer the legs were.

When testing how the robot self-balances in a simulated world, problems occurred with the physics which as a result affected the tests. Whenever the robot experienced events out of the normal such as a push, it immediately would fall down flat without taking into consideration the gravity.

However, the robot was able to maintain self-balance if there were not out of the ordinary events using PID, where the $(P) = 20$, $(I) = 5$, $(D) = 1$, yet again appropriate PID values were tuned after numerous tests with different PID combinations.

6.3 Software simulation

The results from the software simulation were quite different from the physical simulation. After tuning the PID and testing, the robot was able to self-balance when left at peace with no external forces interrupting it such as a push (see Figures 49 - 51). By pushing the robot slightly, it would take it approximately 1 second to regain balance. However, an issue occurred when the robot was pushed with a bit more force because the robot would fall instantaneously on the ground and sometimes would even spin. After analysing the problem the bug turned out to be in the physics of the simulation and since the time was quite limited for the simulation since it was implemented quite late, the bug was not fixed.



Figure 49



Figure 50



Figure 51

As discussed in the previous chapters Reinforcement Learning was partly implemented but not tested in the software simulation as well because of the physics related bug.

6.4 Results

In this section are discussed the results from the sensors, prototypes, and software simulation.

6.4.1 Sensor

Sensor results reflected tremendously on the robot keeping self-balance and the results from the tests were used in order to set goal states and point of no return states. The results from the tests of the initial prototype which were used as a basis afterward showed:

1. Gyroscope
 - Vertical average - $X = 5.7$ and $Y = 7.052$
 - Left average - $X = 6.09$ and $Y = 5.912$
 - Right average - $X = 5.378$ and $Y = 5.946$
2. Accelerometer
 - Vertical average - $X = -0.173$, $Y = 0.05$, $Z = 1.07$
 - Left average - $X = -0.64$, $Y = 0.122$, $Z = 0.829$
 - Right average - $X = 0.66$, $Y = 0.04$, $Z = 0.848$

Please note that as mentioned before, the numbers changed slightly as new changes to the design were made.

For instance, the vertical average numbers (X , Y , Z) were used as a goal states so whenever the robot moves and the sensor reads that the current position of the robot is really close to the goal state. Therefore the signals sent to the motor would be, for example, approximately 6.8 Duty cycle (using MG90S servo motor) which in return would make the motor stop moving. The closer the current values get to either of the two other average points, the higher or lower would the Duty cycle number go, thus the robot would move.

6.4.2 Prototypes

First prototype tests results were implemented using PID and side support from cables (ethernet, power supply, keyboard, mouse, monitor) attached to the Raspberry PI (see Figures 52-54). As seen in the figures, when the robot is pushed, it takes approximately 2-3 seconds to regain balance. When tested using the same optimal PID values and only the power supply cable attached. The results were not that promising due to the reason that the robot kept falling sideways which is one of the reasons to make huge design changes and work on prototype 2.

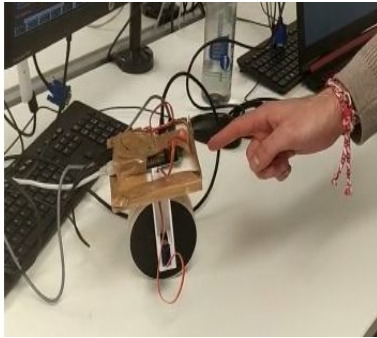


Figure 52

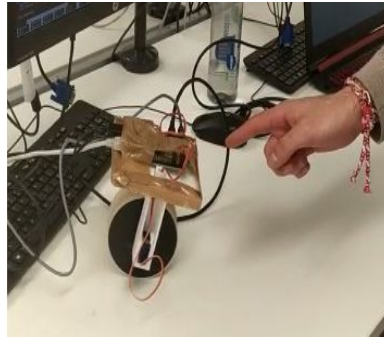


Figure 53

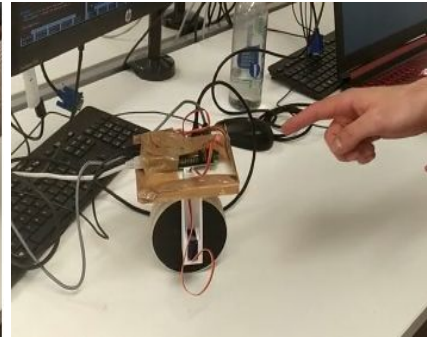


Figure 54

The results from the second prototype were more promising since the robot was not falling sideways anymore, however, two new major issues were found. The first significant issue was that the robot was falling way too rapidly for the motor to respond which is a reason why longer legs were required. By increasing the length of the legs from 9 cm to 21 cm, the results started looking better as the robot had approximately 0.3 seconds more to respond to the changes. However, after additional PID tuning and multiple tests, the robot was still too slow to respond and was still oscillating. Consequently, even longer legs were created and added as an extension, thus the legs went from 21 cm to 30 cm (see Figure 55). Yet again, the PID was re-tuned, and after multiple tests, the result was that the robot was oscillating even less than before, however it could not maintain self-balance for longer than 4 seconds. There was another issue that had been discovered at that particular point of the developing/testing of the robot - the response rate of the motor was causing major issues with the balance because it was too slow.

Once the new PWM based library, which was mentioned in one of the previous chapters, was implemented the tests indicated that the responsiveness of the motor did not improve. Although the newly implemented library is even more low-level based, the results did not change. This meant that a new motor had to be purchased and programmed accordingly, taking into account the responsiveness of the newly purchased motor.



Figure 55

Chapter 7

7.0 Evaluation

The research in Chapter 2 informed the project aims and objectives. The three main objectives of this project - building the robot, self-balance with PID and self-balance with Reinforcement Learning - two of them have been implemented successfully, namely, the robot was built and it manages to self-balance with PID partly. Importantly, going into this particular project, there has always been a risk that a working solution could not be developed given the available resources. As a result, a complete full solution of a self-balancing robot made from scratch was partly developed.

The purpose of this chapter is to evaluate the effectiveness of the two prototypes using PID and how well the robot is able to self-balance. Then compare the results and present data showing for how long the robot can self-balance before losing balance.

Finally, this chapter will finish off by discussing specific areas that can be improved upon. Even though the project did not meet all main goals and functional requirements, a number of improvements which would be advised for a fully working self-balancing robot.

7.1 Self-Balancing

The comparison discussed in this subsection is used to evaluate in a better manner what has been achieved through the development of this project. Since reinforcement learning was not fully implemented and tested, in order to evaluate the progress of making a self-balancing robot was used another approach. By comparing the first and second prototypes using the optimal PID values for each of them correspondingly.

As seen in Figure 56, the angles in degrees ranging from 100 to -100 can be seen on the left of the graph, whereas as at the bottom is shown the time in seconds ranging from 0.0s to 3.0s. The first prototype manages to self-balance for approximately 1.6 seconds, whereas the improved second prototype manages to self-balance for roughly 2.7 seconds. After testing numerous times the most optimal

and effective PID values were found for each of the two prototypes and then both of them were used correspondingly for this evaluation.

The angles degrees are used to display with how many degrees the robot leans forward or backward, where a positive angle degree means that the robot leans forward and a negative means the robot leans backward. Even though on the graph the minimum is displayed as -100, the actual minimum is -90, when the robot has fallen down on its back. Whereas, the maximum is the other end +90, when the robot has fallen down forward.

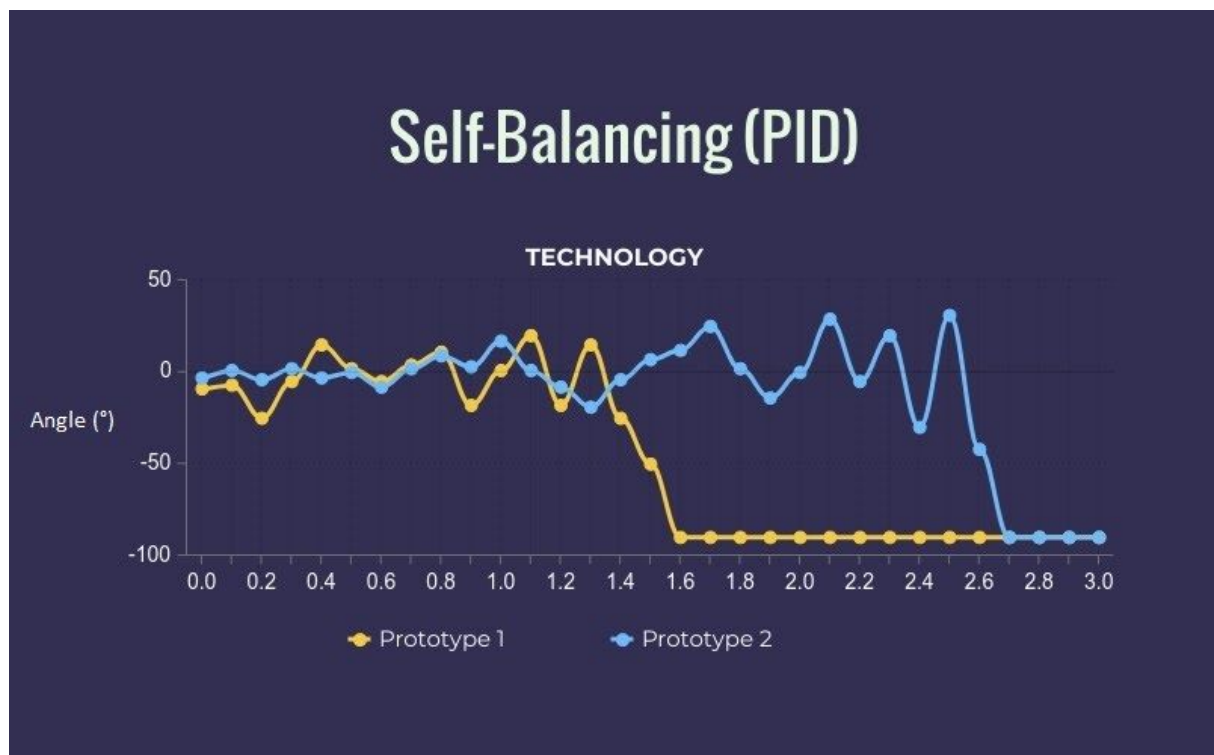


Figure 56

As seen in the Figure 56, the first prototype starts falling with roughly 10 degrees going backward when the signal triggers the motor makes it move backward to regain balance, thus the robot reaches the goal state for a brief at 0.3s and starts falling forward. The robot starts oscillating going back and forth, the degree of oscillating increases until at 1.4s when the motor could not compensate enough and the robot falls down.

On the other hand, the second prototype manages to fix the errors quite well, as its angle degrees are quite close to the goal state (0°). However, at 1s the robot starts oscillating which over time increases. The robot keeps falling backward from 1.0 to 1.3 until it manages to reverse the position of falling. As more time passes and the

oscillating keeps getting greater and greater, the robot starts swinging heavily at roughly 2.0 and finally falls down at 2.7.

The improved design makes the robot fall more slowly compared to its counterpart - the first prototype. This allows more time for the motor to respond and as a result the robot self-balances for longer.

7.2 Legal, Social, ethical and professional issues

Working with hardware has its risk as fire or explosions could be caused, thus damaging the hardware per se or the surroundings including people. Every specific part of the development of the robot has been carefully considered in order to not allow any hardware malfunctioning. Hence, limiting the chance of causing hazardous situations. When testing and evaluating safety measures were taken by ensuring everything is well connected to the correct pins/slots and the cables are not misplaced.

Occasionally, some of the robot's parts' temperature was getting quite high due to the reason that the robot was powered on for prolonged time due to testing and developing. Consequently, the robot would be turned off until the temperature drops down to room temperature. Throughout the whole process of this project's development, the temperature of each of the robot's parts had been checked regularly and tracked in order to minimize any risks.

At all times, when the robot was powered on, it has been taken into consideration that it could misbehave and any harm to its surroundings. Therefore, the robot was never left powered on without me being able to observe how it behaves or powered it off if necessary

As a result, neither the hardware has been damaged, nor any of the surroundings of the robot.

7.3 Improvements

The improvements that could be added to this project in order to make it a more sophisticated robot fall under four areas: robot design, software implementation and hardware functionality, security.

The robot design could be improved further to maximize the effectiveness of the motor on the robot, aesthetics, distribution of weight, etc. The design should be better suited for the hardware and provide

Reinforcement Learning would improve the robot even further as it will cover all the exceptions that could occur in a real-world environment. A combination of a PID

controller and Reinforcement Learning has the potential to make the robot balance itself even more effectively as it would combine two different great approaches for self-balancing.

As tested multiple times in the course of the development of this project, the two servo motors, as easy to control they were, had response rates that were not fast enough and their power should be greater in order to increase the torque.

To control the robot SSH was used which is an approach that has poor security. If an intruder gets access to the robot and the robot is powered on using the batteries, the intruder would be able to move the robot and potentially do harm.

Chapter 8

8.0 Conclusion

This report explored the idea of creating a robot from scratch and making it self-balancing using a PID controller and Reinforcement Learning. After gathering information about hardware, robotics, modeling, and popular practices for building robots, an initial prototype was created which was not able to self-balance well due to design flaws. After multiple iterations a final first prototype started showing better self-balancing results, however, there were still unresolved issues that required a new better design as the robot was not able to self-balance longer than 2 seconds. The second prototype solved many of the issues the initial design had, however, the results were still not promising. Consequently, new design and hardware flaws were found. After multiple iterations most of the newly found design flaws were fixed and the results started looking better, however, the robot was still unable to self-balance for longer than 4 seconds using the optimal values of a PID controller. As the issues that occurred during the design and PID testing stage, the development of Reinforcement Learning was held as it required firstly for the robot to be able to self-balance itself using PID. Single major issues slowed down the whole process of achieving all the goals and objectives of this project, namely, the motors used had too slow responsive rate.

There were multiple challenges involved in order to make the robot self-balancing. After multiple attempts, design changes, electronics changes, motor changes and software adjustments the robot did not manage to keep balance by itself due to the slow responsiveness of the motors. Even though the length of the legs was increased multiple times to up to 30cm which had a substantial difference, the motor was still incapable to react fast enough in order to balance the robot.

The self-balancing robot in the 3D simulated world had more promising results. Being able to self-balance in normal conditions using PID, the robot could not manage to regain balance when pushed by an external force. The reason for not being able to recover stems from a physics bug that was not found and fixed on time. The bug would make the robot fall instantaneously if pushed with a lot of force.

Despite the negative outcomes above, the project defined a solid foundation for future work. Remodeling and changing the motors would be the initial steps to continue improving the robot. The basis of this work can be used as an example and built upon. With a larger pool of knowledge and additional improvements, the robot may one day evolve into something truly helpful for people.

Although there were a number of shortcomings, I view everything in a positive light as the shortcomings highlight how much has been learnt during the development of this project. They offer an opportunity to learn from the mistakes made along the way.

8.1 Critical Appraisal

Overall, I am satisfied with my conduct and the effort I put into this project. I feel I worked to the best of my ability providing the circumstances. The project was quite ambitious and was undiscovered territory for me, an absolute challenge. As a person who has never done anything in Robotics before it challenged my knowledge in multiple fields: Computer Science, Engineering/Designing, Electronics, Physics and Mathematics. Not being able to finish all aims and objectives of this project motivates me even more to learn more about robotics.

For me, the most difficult aspect of the whole project was at the design stage. Designing a robot requires extreme attention to detail and challenges your creativity. It took many design iterations to arrive at the latest design solution and yet there are numerous more design changes that could be made to better the design.

A field that I found interesting before became exceptionally interesting now.

References

- BROOKNER, E., 1998. *Tracking and Kalman Filtering Made Easy*. *Tracking and Kalman Filtering Made Easy*. Available from: www.Wiley.com. [Accessed 6 Nov 2019].
- CHANG, C.L. and CHANG, S.Y., 2017. Using Reinforcement Learning to Achieve Two Wheeled Self Balancing Control. *Proceedings - 2016 International Computer Symposium, ICS 2016*, pp. 104–107.
- DOYA, K., 2010. To cite this article: Kenji Doya (2007) Reinforcement learning: Computational theory and biological mechanisms. *HFSP Journal*, 1(1), pp. 30–40. Available from: <https://doi.org/10.2976/1.2732246/10.2976/1> [Accessed 11 Oct 2019].
- FEINBERG, A., 1996. Markov Decision Processes: Discrete Stochastic Dynamic Programming (Martin L. Puterman). *SIAM Review*, 38(4), pp. 689–689. Available from: https://books.google.co.uk/books?hl=en&lr=&id=VvBjBAAAQBAJ&oi=fnd&pg=PT9&ots=rrmyvGP-NJ&sig=lfj8Dbi2kXmi0nAmTGYyl1vFbCQ&redir_esc=y#v=onepage&q&f=false [Accessed 8 Nov 2019].
- GOODWIN, G.C., GRAEBE, S.F. and SALGADO, M.E., 2007. Control System Design. *IEEE Control Systems*.
- JUANG, H.S. and LURRR, K.Y., 2013. Design and control of a two-wheel self-balancing robot using the arduino microcontroller board. *IEEE International Conference on Control and Automation, ICCA*, pp. 634–639.
- KALMAN, R.E., 1960. (No Title). *Transactions of the ASME-Journal of Basic Engineering*.
- KNOSPE, C., n.d. High-speed magnetically levitated machine tool spindle View project Thrust Magnetic Bearing Modeling and Performance View project. [online]. Available from: <https://www.researchgate.net/publication/3207700> [Accessed 7 Oct 2019].
- PAZ, R.A., 2001. The Design of the PID Controller. *Computer Engineering*.
- SUTTON, R.S. and BARTO, A.G., 1999. Book Reviews Reinforcement Learning. *Journal of Cognitive Neuroscience*, 11(1), pp. 126–134.

TINDER, R.F., 2006. Relativistic flight mechanics and space travel: A primer for students, engineers and scientists. *Synthesis Lectures on Engineering*, 1, pp. 1–112.

WELLING, M., n.d. Reinforcement Learning ICS 273A Instructor: Max Welling. [online]. Available from: <https://www.ics.uci.edu/~welling/teaching/271fall09/RL271-f09.pdf> [Accessed 30 Oct 2019].

ZHONG, J., 2006. PID controller tuning : A short tutorial. *Springer*, pp. 1–13. Available from: <http://saba.kntu.ac.ir/eeed/pcl/download/PIDtutorial.pdf> [Accessed 7 Oct 2019].

Source Code

The source code can be found on github, both the hardware simulation and the software simulation are under the one one github repository. The source code for the hardware simulation could be found in the “RobotScripts” folder, whereas the rest of the files and folders in the repository are used to run the software simulation:

<https://github.com/POBnH4/Autonomous-Robot>

Ethics Form



**ROBERT GORDON
UNIVERSITY•ABERDEEN**

STUDENT PROJECT ETHICAL REVIEW (SPER) FORM

The aim of the University's *Research Ethics Policy* is to establish and promote good ethical practice in the conduct of academic research. The questionnaire is intended to enable researchers to undertake an initial self-assessment of ethical issues in their research. Ethical conduct is not primarily a matter of following fixed rules; it depends on researchers developing a considered, flexible and thoughtful practice.

The questionnaire aims to engage researchers discursively with the ethical dimensions of their work and potential ethical issues, and the main focus of any subsequent review is not to 'approve' or 'disapprove' of a project but to make sure that this process has taken place.

The *Research Ethics Policy* is available at [_ HYPERLINK "http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060" www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060](http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060)

Student Name	Petar Bonchev
Supervisor	Kit-yang Hui
Project Title	Self-balancing Robot
Course of Study	Computer Science
School/Department	CSDM

Part 1 : Descriptive Questions			
1	Does the research involve, or does information in the research relate to:	Yes	No
	(a) individual human subjects		X
	(b) groups (e.g. families, communities, crowds)		X
	(c) organisations		X
	(d) animals?		X
	Please provide further details:		
2	Will the research deal with information which is private or confidential?	Yes	No
			X
	Please provide further details:		

Part 2: The Impact of the Research			
3	In the process of doing the research, is there any potential for harm to be done to, or costs to be imposed on	Yes	No
	(a) research participants?		X
	(b) research subjects?		X
	(c) you, as the researcher?		X
	(d) third parties?		X
	Please state what you believe are the implications of the research:		
4	When the research is complete, could negative consequences follow:	Yes	No
	(a) for research subjects		X
	(b) or elsewhere?		X
	Please state what you believe are the consequences of the research:		

Part 3: Ethical Procedures			
5	Does the research require informed consent or approval from:	Yes	No
	(a) research participants?		X

	(b) research subjects		X
	(c) external bodies		X
	If you answered yes to any of the above, please explain your answer:		
6	Are there reasons why research subjects may need safeguards or protection?	Yes	No
			X
	If you answered yes to the above, please state the reasons and indicate the measures to be		
7	Has PVG membership status been considered?		X
	(a) PVG membership is not required.		X
	(b) PVG membership is required for working with children.		X
	(c) PVG membership is required for working with protected adults.		X
	(d) PVG membership is required for working with both children and protected		X
	If you answered yes to (b), (c) or (d) above, please give details:		
8	Are specified procedures or safeguards required for recording, management, or storage of data?	Yes	No
			X
	If you answered yes to the above, please outline the likely undertakings:		

Part 4: The Research Relationship			
9	Does the research require you to give or make undertakings to research participants or subjects about the use of data?	Yes	No
			X
	If you answered yes to the above, please outline the likely undertakings:		
10	Is the research likely to be affected by the relationship with a sponsor, funder or employer?	Yes	No
			X
	If you answered yes to the above, please identify how the research may be affected:		

Part 5: Other Issues			
----------------------	--	--	--

11	Are there any other ethical issues not covered by this form which you believe you should raise?	Yes	No
			X

Statement by Student			
I believe that the information I have given in this form is correct, and that I have addressed the ethical issues as fully as possible at this stage.			
Signature	Petar Bonchev	Date	02/10/2019

If any ethical issues arise during the course of the research, students should complete a further Student Project Ethical Review (SPER) form.

The *Research Ethics Policy* is available at [_ HYPERLINK "http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060" www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060](http://www.intranet.rgu.ac.uk/credo/staff/page.cfm?pge=7060)

Part 6: To be completed by the supervisor			
12	Does the research have potentially negative implications for the University?	Yes	No
			✓
If you answered yes to the above, please explain your answer:			
13	Are any potential conflicts of interest likely to arise in the course of the research?	Yes	No
			✓
If you answered yes to the above, please identify the potential conflicts:			
14	Are you satisfied that the student has engaged adequately with the ethical implications of the work? [In signifying agreement, supervisors are accepting part of the ethical responsibility for the project]	Yes	No
		✓	
If you answered no to the above, please identify the potential issues:			
15	Appraisal: Please select one of the following		
	The research project should proceed in its present form – no further action is required	✓	
	The research project requires ethical approval by the School Ethics Review Panel		
	The research project needs to be returned to the student for modification prior to further action		
	The research project requires ethical review by an external body. If this applies please give details		

	Title of External Body providing ethical review	
	Address of External Body	
	Anticipated date when External Body may consider project	

Affirmation by Supervisor			
I have read the student's responses and have discussed ethical issues arising with the student. I can confirm that, to the best of my understanding, the information presented by the student is correct and appropriate to allow an informed judgement on whether further ethical approval is required.			
Signature	K. Hui	Date	02/10/2019