UNIVERSITY OF Southampton

Theory, Design and Building of a Bipedal Robot Based on the Common Quail

FEEG3003: Individual Project

Ryan Khoo Yeap Hong

30480183

Supervised by Dr Suleiman Sharkh

April 2022

Word Count: ~10,850

This report is submitted in partial fulfilment of the requirements of the MEng Mechanical Engineering, Faculty of Engineering and the Environment, University of Southampton.

Abstract

Robots are vital to the functioning and automation of many modern processes. Many of these robots are only able to perform their functions within the environments designed around them. A better approach may be a robot designed around its environment Instead. As most environments are already designed for bipedal humans, bipedal robots are uniquely poised to fill this niche. Therefore, this report aims to explore and develop a bipedal robot capable of semi-autonomous walking.

A review of the literature surrounding bipedal robotics is performed and finds that bird-based configurations may provide a better basis for the development of bipedal platforms compared to human-based approaches due to lower centres of mass providing better stability.

Pavo, a 30cm tall robot inspired by the common quail (*Coturnix Coturnix*) is built with off-the-shelf parts and 3D printing to reduce weight and costs. A novel control theory utilising fuzzy logic is developed and implemented to reduce the computational power required. An Arduino UNO is used to realise this control theory and execute the dynamic footsteps required to enact balanced walking.

Various complications are encountered throughout its development and are remedied or future improvements suggested. Issues such as underpowered servos prevented a faithful recreation of a quail gait cycle and full realisation of the fuzzy logic system. However, Pavo is still able to demonstrate appropriate responses to external stimuli and ultimately shuffle forward at a speed of 29.371mm/s.

Table of Contents

Abstract 1

Table of Contents 2

Declaration 4

Acknowledgements 5

Abbreviations, Figures and Tables 6

1. Introduction 10

1.1. History and Literature Review 11

1.2. Project Aims and Objectives 19

2. Design and Building 20

2.1. Design Background 20

2.2. Hardware/Components 22

2.3. Manufacture Details and Specifications 23

3. Control Theory and Programming 31

3.1. General Approach and Overview 31

3.2. Controller Code Theory/Bluetooth Communication 32

3.3. Fuzzy Logic 33

3.4. Code Details 36

3.4.1. Ultrasonic Sensor Implementation 36

3.4.2. IMU Implementation 37

3.4.3. Fuzzy Logic Implementation 37

3.4.4. Servo Set Structure 38

3.4.5. Cycle Stages 39

3.4.6. Cycle Stage Control 41

3.4.7. Direction Defining and New Position Generation 42

3.4.8. Eased Position Transitions 43

3.4.9. Pending Set Container and Periodic Position Execution 45

4. Results and Discussion 47

4.1. Post-Assembly Adjustments 47
4.1.1. SRAM Limitations 47
4.1.2. Servo Torque Limitations 47
4.2. Results 49
4.3. Issues, Improvements and Future Work 53
4.3.1. Servo Torques and Power 53
4.3.2. Improved Construction of the Robot 53
4.3.3. Computing Power 54
4.3.4. Bluetooth Communication 56
4.3.5. Fuzzy Logic 56
4.3.6. Environmental Data/Feedback 56
4.3.7. Gait/Walking Cycle 57
5. Conclusion 58
References 59
Appendix 64
A.1 Video Figure Links 64
A.2 Parts list and Costing 64
A.3 Software Utilised 65
A.4 Risk Assessment 66
A.4.1 Risk Assessment Forms 67
A.4.1.1 Design workshop Method Statement 67
A.4.1.2 Design workshop Risk Assessment 68
A.4.1.3 Electronics workshop Method Statement 71
A.4.1.4 Electronics workshop Risk Assessment 72
A.5 Full Controller State Commands 74
A.6 MATLAB Code 75
A.7 Speed Test data 75
A.8 Arduino Code: 76
A.8.1 Controller Code 76
A.8.2 Robot Code 79
A.9 Engineering Assembly Drawings 100

Declaration

I, Ryan Khoo Yeap Hong declare that this thesis and the work presented in it are my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a degree at this University;

2. Where any part of this thesis has previously been submitted for any other qualification at this University or any other institution, this has been clearly stated;

3. Where I have consulted the published work of others, this is always clearly attributed;

4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;

5. I have acknowledged all main sources of help;

6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;

7. None of this work has been published before submission.

Acknowledgements

I would like to extend my sincere thanks to my supervisor, Dr Suleiman Sharkh, for his continued and substantial support provided throughout this project, his guidance and feedback were invaluable in determining the direction of the project and in overcoming roadblocks/hardships faced.

I would also like to express my gratitude to the technicians and persons in charge of the design and electronics workshops at the University of Southampton for enabling the manufacture and creation of the final product.

I would also like to recognise the many individuals behind the literature referenced in this work, and the countless hours of research and collaboration that have led to the current state of the art, without which the project would not have been possible.

And finally, I would like to acknowledge friends and family for providing their unwavering moral support throughout the project.

Abbreviations, Figures and Tables

Abbreviations:

CoM	-	Centre of Mass
CoP	-	Centre of Pressure
DoF	-	Degrees of Freedom
SLIP	-	Spring-Loaded Inverted Pendulum
ZMP	-	Zero Moment Point
DS	-	Double Limb Support
SS	-	Single Limb Support
IMU	-	Inertial Measurement Unit
SEA	-	Series Elastic Actuators
BLDC	-	Geared Brushless Direct Current (Motors)
LIDAR	-	Light Detection and Ranging
CNC	-	Computer Numerical Control
LED	-	Light Emitting Diode
SRAM	-	Static Random-Access Memory
GPS	-	Global Positioning System
CPG	-	Central Pattern Generator
GD	-	Generative Design
PLA	-	Polylactic Acid
DMLS	-	Direct Metal Laser Sintering

Figures:

Figure 1: An Ocado “swarm” packing warehouse with wheeled robots (Grylls 2018) 10
Figure 2: A breakdown of the human walking gait (Lohman et al. 2011) 12
Figure 3: “WABOT-1” Bipedal robot by Waseda University (Takanishi 2019).....13
Figure 4: Left) Boston Dynamics’ “Atlas”, right) Agility Robotics’ “Cassie” (Ficht and Behnke 2021; Reher, Ma, and Ames 2019) 14
Figure 5: Left) The NimbRo-OP2X robot, Right) Off-the-shelf servos in a 3D printed assembly make up the robot’s hip joints (Ficht et al. 2018) 16
Figure 6: A full gait cycle of a common quail, experimentally obtained by Abourachid et al. (2011) utilising high-speed video fluoroscopic recordings.....18
Figure 7: A DoF diagram illustrating the placements of the servos with a centre “spine” (in red) suspended between the two legs. Lengths obtained by Lepora et al. (2016) and planes of movement possible for each servo labelled 21
Figure 8: The simplified damped spring inverted pendulum model with pivot point and CoM labelled.....21
Figure 9: a) An Arduino UNO, b) A Raspberry Pi (Carolo 2020).....22
Figure 10: a) Adafruit PCA9685 16-Channel Servo Driver (Earl 2012), b) Adafruit TDK InvenSense ICM-20948 9-DoF IMU (Siepert 2020), c) SER0056 Servo, d) HC-SR04 ultrasonic distance sensor, e) Nintendo “nunchuck” controller, f) Spring damper suspension, g) HC-05 Bluetooth serial transceiver module 23
Figure 11: a) overall view of Pavo with the axis sign conventions utilised in the remaining report, b) Front view of Pavo, c) Top view of Pavo, d) Side view of Pavo.....24
Figure 12: A side view of the finished robot.....25
Figure 13: A labelled circuit diagram illustrating the connections between components onboard Pavo 26
Figure 14: The “head” of Pavo 26
Figure 15: a) A top view of the hip, embedded IMU and servo driver b) A bottom view of the hip.....27
Figure 16: The “tail” of Pavo, housing the batteries and a power switch, both labelled...27
Figure 17: The middle section of Pavo with the spine joint and servos labelled.....28
Figure 18: The left foot assembly of Pavo with spring damper labelled 28
Figure 19: The foot of Pavo with the distances between the three contact points marked 29
Figure 20: The finished controller with remote.....30
Figure 21: A labelled circuit diagram of the electronics in the controller 30
Figure 22: A high-level diagram illustrating the flow and functions of the control system designed.....32

Figure 23: Membership functions of crisp angle input 34
Figure 24: Membership functions of crisp angular velocity input 35
Figure 25: Output membership function plots utilising four fuzzy output variables..... 35
Figure 26: An example of the fuzzy logic system in use with normalised inputs (Angle = 5, Angular velocity = -24), and resulting normalised crisp output of -13.6 36
Figure 27: The resulting fuzzy logic surface plot with crisp outputs on the vertical Z-axis and crisp inputs of angle and angular velocity on the X and Y-axis..... 36
Figure 28: The raw roll value (blue) and the filtered, smoothed roll value (red) 38
Figure 29: The raw Y-axis gyro value (blue) and the filtered value (red), with the filter effectively extracting a steady rhythm out of a noisy input resulting from the gait cycle of Pavo 38
Figure 30: The structure of a position set utilised in the code, with nine servo values and a tenth “divisions” value that determines the number of transitionary values to be generated 39
Figure 31: Parallel number lines illustrating the mapping of true servo angles to their normalised values utilised in the position sets with an example of a set value of 20 resulting in a servo angle of 139°..... 39
Figure 32: The resulting step pattern of eight “empty” sets combined with their eight corresponding “front/back” sets, with the ground contacts indicated by a red “X”..... 41
Figure 33: A diagrammatic representation of the cyclic stage variable system..... 42
Figure 34: A graph illustrating various directions and magnitudes of movement derived from the X and Y modifiers, with the vertical X-axis representing walking forward/back relative to Pavo (pictured from above) 43
Figure 35: The three possible transitionary trends, a) Linear, b) Speeding up, c) Slowing down, with an x-axis of time and y-axis of servo position, 0 representing the old state and 100 the new target state 44
Figure 36: Format of “pending sets” container array, with “x” denoting a servo position value..... 45
Figure 37 A control diagram detailing how all the individual functions detailed above in section 3 work together to imitate a CPG..... 46
Figure 38: Two views of the added rubber bands (blue) attached to the knee to aid the servos, indicated by red arrows 48
Figure 39: A force/moment diagram illustrating the rubber band modification (blue) counteracting weight induced moments..... 48
Figure 40: [VIDEO] A demonstration of the cyclic footstep functionality with slowed down and exaggerated movements for a) walking front and b) stepping right (full links can be found in the appendix (A.1) if embedded links are not functioning) 49

Figure 41: [VIDEO] A demonstration of Pavo's ability to respond accordingly to angular position and come to rest when rebalanced a) Tilting forwards and walking forward, b) Tilting backwards and walking backwards, c) Tilting right and walking right, d) Tilting left and walking left 50
Figure 42: [VIDEO] A demonstration of Pavo's ability to respond to a sensed obstacle and begin a backwards movement and stop when at a safe distance 50
Figure 43: [VIDEO] A demonstration of Pavo's ability to respond to controller input appropriately 51
Figure 44: [VIDEO] A demonstration of Pavo walking 70cm 51
Figure 45: [VIDEO] Six speed tests overlayed 52
Figure 46: [VIDEO] Five directional tests overlayed 52
Figure 47: A generatively designed calf prosthetic, reducing material cost and weight while maintaining required strength (Rajput et al. 2021) 54
Figure 48: A size comparison of: a) Arduino DUE, b) The currently implemented Arduino UNO, c) Raspberry Pi (Senese 2012) 55

Tables:

Table 1: A table illustrating the possible commands that are sent over Bluetooth 33
Table 2: A table showing the contact of each foot with the ground during the transitions between the eight states, and the resulting single support (SS) or double support (DS) state of a stage, with ground contact states indicated in grey 40
Table 3: A table describing the eight parts of each of the three sets, with "L" denoting the left leg and "R" denoting the right leg. C1 represents the first contact of the foot with the ground, C2, the second, and so on, until the leg is lifted during the swing phase, indicated by "air" 40

1. Introduction

The field of robotics has seen great strides in the past century. With the concept of useful man-made automata jumping from the pages of early 20th-century science fiction into very real, practical applications today (Carlos De Pina Filho and dos Santos Mota 2010). Robots, in all their current forms, are ubiquitous in modern society, bringing unprecedented automation into countless industries such as the automotive sector (Bartoš et al. 2021).

A vital aspect of many of these robots is their mobility, such as to navigate a factory floor or warehouse. In most industrial applications, these robots have been developed in tandem with the environment they operate in, with their surroundings fitted with flat floors, tracks, markers, etc. Such as seen in Figure 1; a highly automated packing warehouse where neither wheeled robot nor environment can be used independently.

Figure 1: An Ocado "swarm" packing warehouse with wheeled robots (Grylls 2018)

While systems such as these have been implemented with great success, they require large investments, long construction times, and must be fully conceptualised from an early stage of development. These highly-specialised robots are limited to the environments designed around them, and will never be employed in the countless pre-existing environments designed around humans.

It may then be argued that robots designed around their environments instead would be a more suitable approach, providing backwards compatibility that allows them to be readily integrated into existing environments designed around bipedal humans.

Therefore, a mobility solution that satisfies this design philosophy is bipedal robots. Sufficiently advanced, bipedal robots can navigate the majority of environments that

humans currently operate in, doing so better than wheeled, threaded or even quadrupedal robots. They are able to, for example, climb stairs, and navigate uneven surfaces and tight areas that other mobility solutions cannot.

1.1. History and Literature Review

To design bipedal robots, it is beneficial to first study and understand how this form of movement is achieved in nature. Bipedalism in nature has been finetuned over millennia to suit a specific organism's needs and environments, be it for improved energy efficiency, reaching higher places or freeing up other appendages for flying/tool-use, this provides a strong foundation for engineers to develop upon (Hunt 2015; Lepora et al. 2016).

The first, most obvious organism to study for the development of bipedal robots would be ourselves; Homo Sapiens. Much work has been done breaking down and classifying the various movement patterns of a generic human gait (Vaughan 2003). Many styles of movement such as running or skipping can be achieved with this bipedal configuration. However, this report will place a primary focus on walking as it is a logical starting point for the development of bipedal robots.

Walking is defined by Linden (2011) as *"a repetitious sequence of limb motions to simultaneously move the body forward while also maintaining stance stability"* (van der Linden 2011). A human walking cycle can be viewed as a series of motions punctuated by distinct stages/positionings. Muscles use energy to both reposition the components of the legs and accelerate/decelerate the masses involved (Hunt 2015). It is also important to note that while each cycle is repetitive and similar, they are not the same in practice, with minute adjustments applied to each step to maintain the system and avoid falling. The human walk cycle can be broken down into two distinct stages; the stance and swing stages, as can be seen in Figure 2 below by Lohman, Balan Sackiriyas, and Swen (2011).

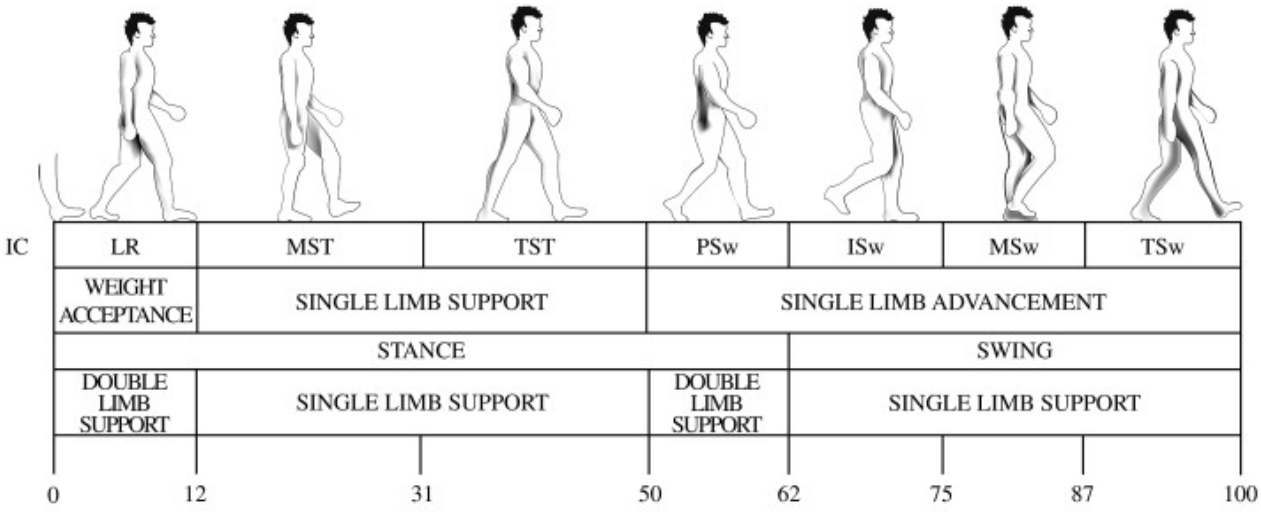


Figure 2: A breakdown of the human walking gait (Lohman et al. 2011)

Beginning with both feet in contact with the ground, known as double limb support (DS), the “stance” stage encompasses the motions from initial DS, swinging a foot forward, and resuming DS on contact with the ground. The “swing” phase then begins upon single limb support (SS) once the other foot leaves the ground, and ends upon the re-establishing of DS, this then begins a new “stance” stage and the cycle is repeated (Liu, Chen, and Chen 2019; Lohman et al. 2011).

When considering the dynamics of the entire system during this cycle, it exhibits properties similar to that of an inverted pendulum (Hunt 2015), which systems of bipedal robot control can be based upon (Liu and Qian 2019), further discussed in this section below.

As discussed in the introduction, bipedal robots have been in the collective consciousness of scientists and researchers for many decades (Carlos De Pina Filho et al. 2010). However, bipedal robotics as we know it today only began development in earnest in the late 1960s, with studies by researchers such as R.B. McGee beginning work on theories of legged locomotion and algorithms capable of coordinating leg movements (McGhee 1968). Completed in 1973, the WABOT-1, a 1.5-meter-tall robot by Japan-based Waseda University was developed (seen in Figure 3), capable of emulating a pre-programmed human gait, and is generally regarded as the first modern bipedal robot (Bruemmer and Swinson 2003; Takanishi 2019; Vaughan 2003).

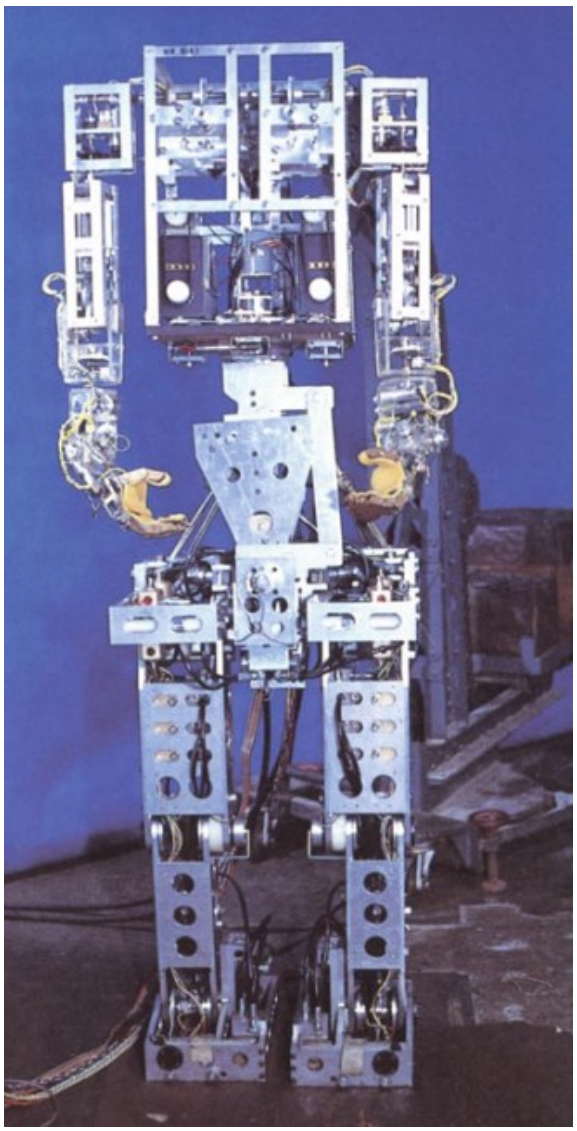


Figure 3: "WABOT-1" Bipedal robot by Waseda University (Takanishi 2019)

Since then, substantial progress has been made in the development of bipedal robots, culminating in the present state of the art, with robots such as Boston Dynamics' Atlas and Agility Robotics' Cassie bipeds (Figure 4) showcasing some of the most sophisticated implementations of bipedal robots thus far. Despite the tremendous work that has been done, the field is far from stagnation and may still be considered relatively new (Ficht and Behnke 2021; Liu et al. 2019).

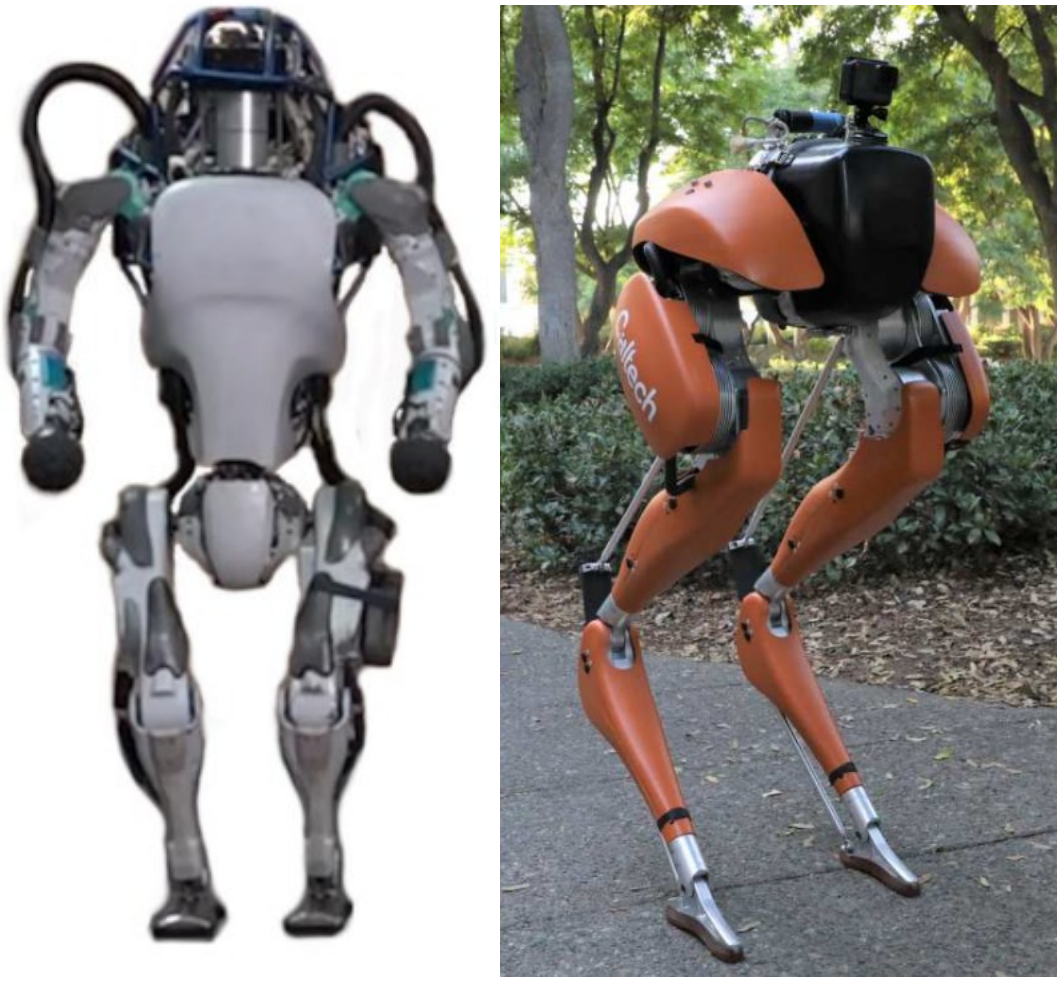


Figure 4: Left) Boston Dynamics' "Atlas", right) Agility Robotics' "Cassie" ***(Ficht and Behnke 2021; Reher, Ma, and Ames 2019)***

While more challenging to develop and control, there is sufficient incentive behind the development of these machines. As discussed above, a primary motivation is that bipedal robots are potentially better suited to traverse terrain that more traditional movement solutions such as wheels or threads cannot (Warnakulasooriya et al. 2012). Sufficiently developed, bipedal robots would be able to traverse both natural and urban, human-centric environments far more efficiently than any traditional solution. This advantage would manifest itself in meaningful applications such as in hospitals, emergency response, replacing humans in dangerous tasks or monitoring locations far from human habitation (Carlos and Pina 2010; Liu et al. 2019; Xie et al. 2020). Bipedal robots may also find a place in the domestic/civilian markets, with implementations such as entertainment, delivery, waiting/hotel staff, and medical/household helpers. Developments in the field may even find their way into other industries such as medical exo-skeletal aids (Aithal et al. 2021; Liu et al. 2019; Reis et al. 2020).

Many different configurations have been utilised to create bipedal robots, with the average speed and size of platforms increasing over the years as underlying technologies constantly improve, enabling bigger, better robots to be constructed. Some designs only incorporate legs, while others emulate entire humanoid forms. These robots also vary in their use of biomimicry, with some utilising the bare minimum degrees of freedom (DoFs) to realise bipedal movement, while others attempt to rival the whopping 30 DoFs present in a single human leg (Aithal et al. 2021; Xie et al. 2020).

Among the most important components in these robots are their actuators. Many solutions exist and have been implemented. In earlier years, electrical actuators with high ratio reducers were a common choice, offering a good balance between speed, torque and size. As the field progressed and compliance increasingly deemed an important aspect of bipedal robots (compliance further discussed below), alternative solutions such as series elastic actuators (SEAs) were utilised in robots such as NASA's Valkyrie platform. Other solutions have also been employed such as geared brushless DC (BLDC) motors for their small form factor and ability to provide torque feedback via current sensing. These mostly electrical solutions are however not perfect, being susceptible to overheating and other issues. Where these were deemed too detrimental, hydraulics has also been implemented into bipedal systems and has seen success in implementations such as Boston Dynamics' Atlas line, although not without their drawbacks such as leaks, noise and weight (Ficht and Behnke 2021).

Just as important are the feedback systems in place to allow for the system to respond to and interact with its environments. A vital sensory component found in the majority of bipedal robots is the inertial measurement unit (IMU), providing acceleration, position and gyroscopic feedback to allow the system to predict its trajectories and compensate accordingly to prevent falls. Another common feedback utilised is joint positioning data, allowing the system to know exactly how it is configured at any one time, this data may be gathered via highly accurate encoders or more traditional potentiometer-based solutions. Other more niche forms of feedback are utilised by select robots, such as light detection and ranging (LIDAR) systems or camera systems like those used in Honda's ASIMO (Carlos De Pina Filho et al. 2010; Ficht and Behnke 2021).

The materials and manufacturing of these robots are also important. With common design requirements such as high strength and low weight, metal alloys are commonly used. These bipedal research platforms are highly unique and usually produced at low volume, therefore, manufacturing methods such as computer numerical control (CNC) milling and 3D printing are also commonly utilised. Recent 3D printing developments have made it a particularly well-suited manufacturing method for these highly specialised applications, enabling high complexity with no extra cost, and reducing the start-up/tooling costs

characteristic of more traditional manufacturing methods. Components are also increasingly being bought off-the-shelf to decrease costs, such as in the NimbRo-OP2X robot, where off-the-shelf servos were utilised with 3D printing (pictured in Figure 5) to achieve a low cost compared to the prohibitively high prices of most bipedal robot research platforms (Ficht et al. 2018; Ficht and Behnke 2021).

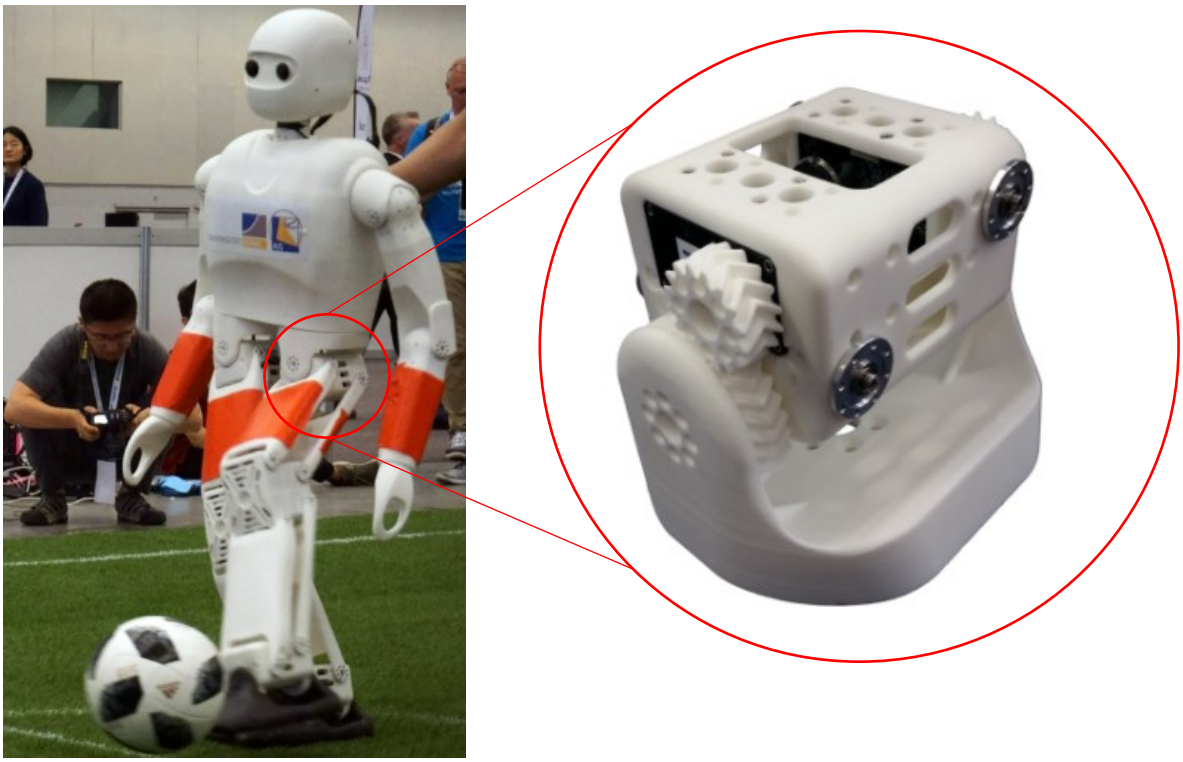


Figure 5: Left) The NimbRo-OP2X robot, Right) Off-the-shelf servos in a 3D printed assembly make up the robot's hip joints (Ficht et al. 2018)

Compliance in bipedal designs is becoming increasingly important as more research is conducted. Components such as springs and pulleys decrease forces and torque spikes on joints and store energy much like ligaments and tendons do in living organisms (Rajput et al. 2021), they have been researched/implemented in projects such as in Badri-Spröwitz et al. (2022), Maiorino and Muscolo (2020), Park et al. (2011) and Tsagarakis et al. (2017).

Throughout the years, many control strategies have been developed. As mentioned above, one of the oldest and most pervasive control theories in the field are models like the spring-loaded inverted pendulum (SLIP) method based on the simplification of the robot into a single inverted pendulum pivoting about a zero-moment-point (ZMP) or centre-of-pressure (CoP), as used by Nguyen et al. (2020), Bae and Oh (2018), Chang et al. (2020) and Lin et al. (2021). These control theories have also been improved and built upon, such as adding a damping aspect or considering a double inverted pendulum as a better approximation of a full humanoid configuration, but an issue remains; a complex

real-world robot cannot be reliably simplified into something as abstract and simplistic as a pendulum about a perfect pivot point.

This has been identified and calls for a different approach, a “model-free control design” have been made (Hu and Smith 2000). While there have been a few approaches developed based on this model free design, in this report, a novel fuzzy logic-based approach was considered.

Fuzzy logic allows the processing of data that would not otherwise be feasible with more traditional control methods. It does this in a much more naturalistic way, mimicking human cognition (Singh et al. 2013). The theory is based on relative membership functions, where “crisp”, numerical inputs are “fuzzified” into fuzzy variables via set membership functions and used in an intuitive inference system that manipulates the variables according to a set of human-comprehensible rules. These fuzzy outputs are then “defuzzified” by output membership functions and return a usable numerical output (Douglas 2021; Zadeh 1965), further detailed in section 3.3 below.

Humans are not the only bipedal organisms to inhabit the earth, more than then-thousand species of bird, and by extension, another ten thousand of their extinct theropod ancestors are bipedal (Brusatte, O’Connor, and Jarvis 2015; Lepora et al. 2016). The earliest signs of bipedalism in mammals may be placed at 3.7 million years ago, with fossilised hominid footprint patterns and fossils such as the *A. afarensis* “Lucy” strongly indicative of mammalian bipedalism. (Finlayson 2005; Hunt 2015; Vaughan 2003). However, the fossil record also contains specimens displaying bipedalism in birds, with fossils such as the *Archaeopteryx* and various small bipedal theropods dated over 150-200 million years ago. This 150-million-year evolutionary lead may very well have resulted in musculoskeletal configurations better suited for bipedalism (Brusatte et al. 2015; Ksepka 2022; Lepora et al. 2016).

With this in mind, emphasis is placed on alternative, non-human gaits and designs. Birds are incredibly varied and exhibit many different forms of bipedal movement such as hopping, striding, running and skipping. However, the walking gait of a quail (*Coturnix Coturnix*) is reviewed as quails tend to fly less and are more well-adapted for walking than most birds (Abourachid et al. 2011), this can be seen in Figure 6.

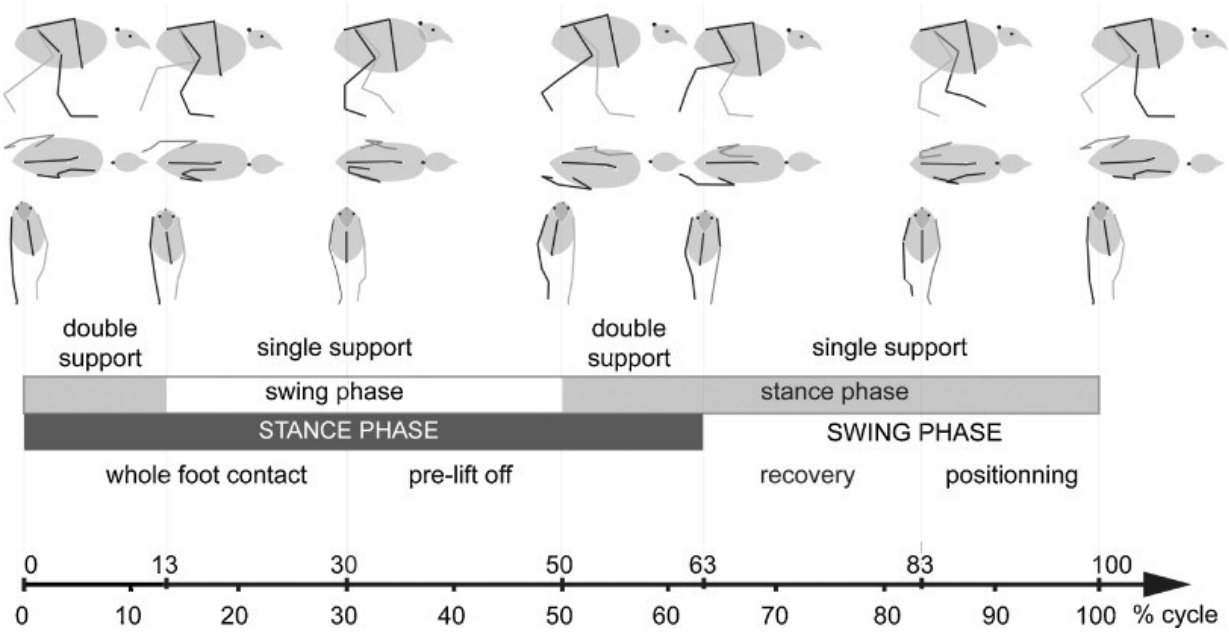


Figure 6: A full gait cycle of a common quail, experimentally obtained by Abourachid et al. (2011) utilising high-speed video fluoroscopic recordings

From Figure 6, it can be seen that when analysed in the same way as a human gait cycle (as in Figure 2), the gait patterns are near identical, with a stance phase difference of only 1% of a total cycle; a potential example of evolutionary convergence. The only remaining difference is therefore the characteristic “crouched” build of birds resulting in a much lower CoM and improved inherent stability over that of a walking human as concluded by Lepora et al. (2016) and Andrada et al. (2014).

1.2. Project Aims and Objectives

Against this background, the project aims to develop a walking robot designed around human environments by utilising a quail-based bipedal configuration. A complex engineering problem such as this comprises many facets. Therefore, the objective of the project is to test and fulfil the following design specifications that the above aim may be divided into:

- To build and manufacture a lightweight, robust bipedal robot whose design balances biomimicry and practicality.
- To develop and code a reliable control system capable of generating dynamic, adaptive footsteps to drive the servos on the robot and enable walking.
- To ensure a strong foundation is designed, both for the code and physical components that allow for upgradability and future developments.
- To explore the use and implementation of a lightweight control theory approach using fuzzy logic to replace traditional, more computationally expensive control theories.
- To build the robot with components totalling under approximately £100.

2. Design and Building

2.1. Design Background

Based on the literature above, an avian-inspired design was ultimately utilised based on the measurements and configuration of a common quail (*Coturnix Coturnix*) recorded by Lepora et al. (2016) and supplemented by Nyakatura et al. (2012). These measurements informed the relative distances between joints and the overall configuration of components.

This was implemented in the form of a central “spine” assembly, suspended between the legs meant to represent the mass of all non-leg parts of the quail, it can be manipulated via servos and houses the Arduino, Bluetooth communication module, ultrasonic sensor, and batteries. This system, independent of the legs, aims to allow Pavo to balance and counteract disturbances resulting from leg swings.

To minimise costs, the minimum number of actuators required to realise an approximation of the quail walking cycle (in Figure 6 above) is utilised. Two axes of movement are present at the hip, one across the sagittal plane, and one across the frontal plane. The knee joint is allowed to pivot in the sagittal plane and three servos were utilised to manipulate the spine relative to the rest of the body, allowing it to pitch up and down, roll side to side, and yaw left and right. This resulted in nine total DoFs, requiring nine servos. This configuration can be seen in Figure 7 below. The servos utilised have relatively low torque outputs, resulting in size and weight limitations further detailed in sections 2.3 and 4.2 below.

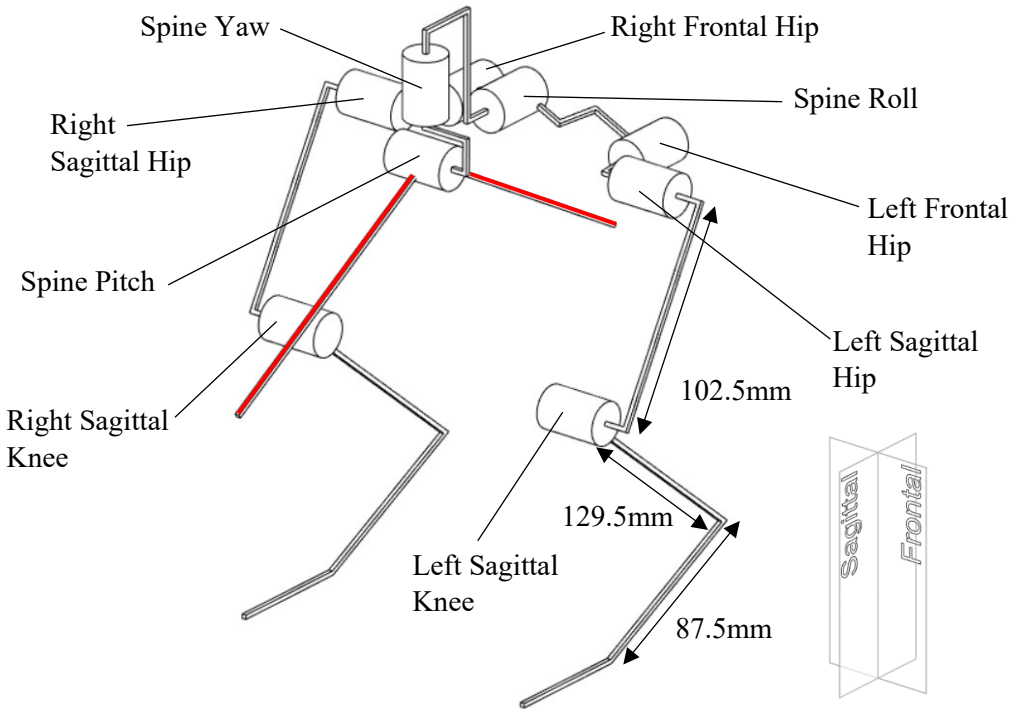


Figure 7: A DoF diagram illustrating the placements of the servos with a centre "spine" (in red) suspended between the two legs. Lengths obtained by Lepora et al. (2016) and planes of movement possible for each servo labelled

This approximation of a quail's skeletal configuration notably removes the active actuation of the joint connecting the tibiotarsus and tarsometatarsus (the ankle joint). This was replaced with a passive damped spring to reduce bounce during steps.

Despite not being utilised for calculations/modelling, a representation of the equivalent system during a single support phase can be seen in Figure 8 below to illustrate the simplification utilised in more traditional control theories, a far cry from the complexity of the system seen in Figure 7.

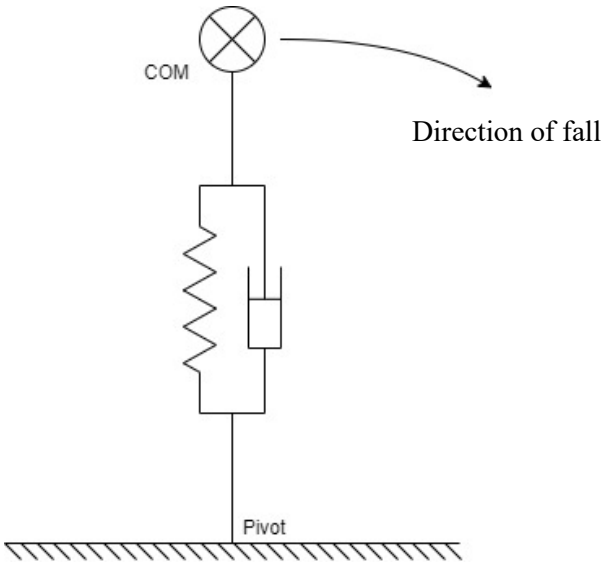


Figure 8: The simplified damped spring inverted pendulum model with pivot point and CoM labelled

As in the abstract, the robot has been named “Pavo”, a homage to the “*Pavoninae*” subfamily of birds to which the quail belongs, and will be referred to as such for the remainder of the report.

2.2. Hardware/Components

Many off-the-shelf parts were utilised in the construction of Pavo as they lower costs and reduce the time and work required to build the robot, these components are described below (Carlos De Pina Filho et al. 2010; Ficht and Behnke 2021).

Two processor solutions were considered (an Arduino-style microprocessor and a Raspberry pi computer) as they were both adequately sized (seen in Figure 9) and deemed capable of performing the tasks required. These were compared, and the Arduino UNO was ultimately chosen for the following reasons:

- Simpler to code and implement
- Near-instant start-up and power down
- Raspberry Pi computational power was deemed excessive for the use case
- Arduinos are more power-efficient
- Arduinos style boards are generally cheaper

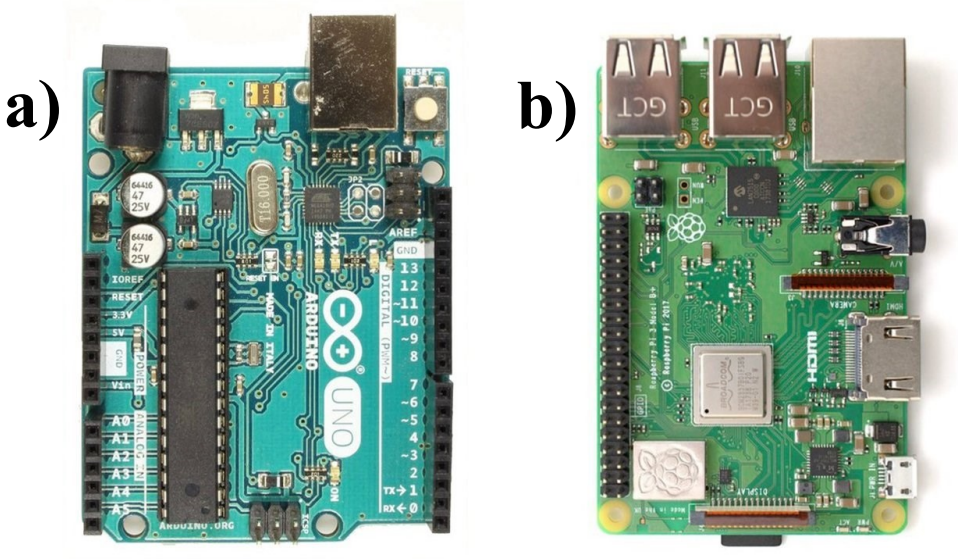


Figure 9: a) An Arduino UNO, b) A Raspberry Pi (Carolo 2020)

In addition to the Arduino UNO processor, various other notable components were utilised in the construction of Pavo, such as Bluetooth modules to facilitate wireless communication between the controller and the onboard Arduino to prevent a wire tether from interfering with Pavo’s movements. These can be seen in Figure 10 below.

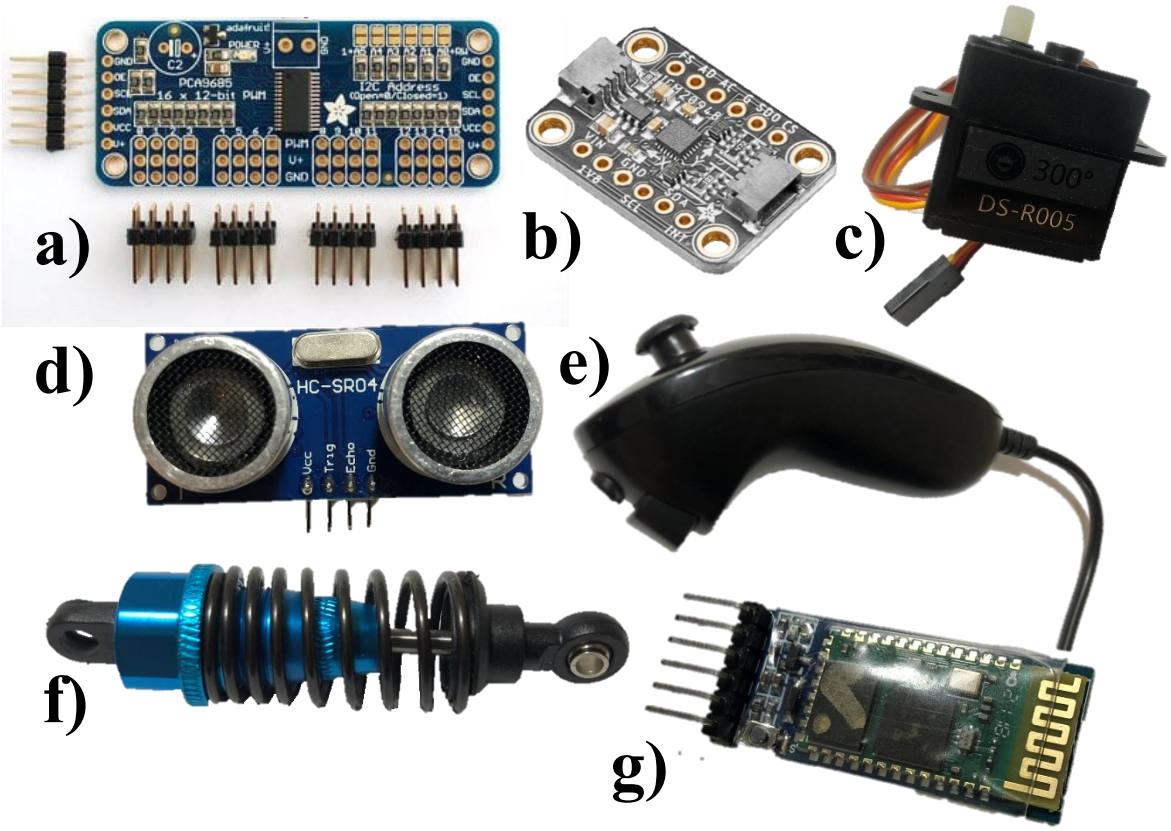


Figure 10: a) Adafruit PCA9685 16-Channel Servo Driver (Earl 2012), b) Adafruit TDK InvenSense ICM-20948 9-DoF IMU (Siepert 2020), c) SER0056 Servo, d) HC-SR04 ultrasonic distance sensor, e) Nintendo "nunchuck" controller, f) Spring damper suspension, g) HC-05 Bluetooth serial transceiver module

This resulted in Pavo costing just over £120. A detailed breakdown of cost, with specifications and sources for each component, can be found in the appendix (A.2) along with a full list of software utilised in the project (A.3).

2.3. Manufacture Details and Specifications

With these components and design parameters in mind, the full robot was designed in Autodesk Fusion 360 and tested with virtual joints to ensure components would not collide during its operation. An engineering assembly drawing for Pavo can be found in the appendix (A.9). The final design can be seen in Figure 11 below.

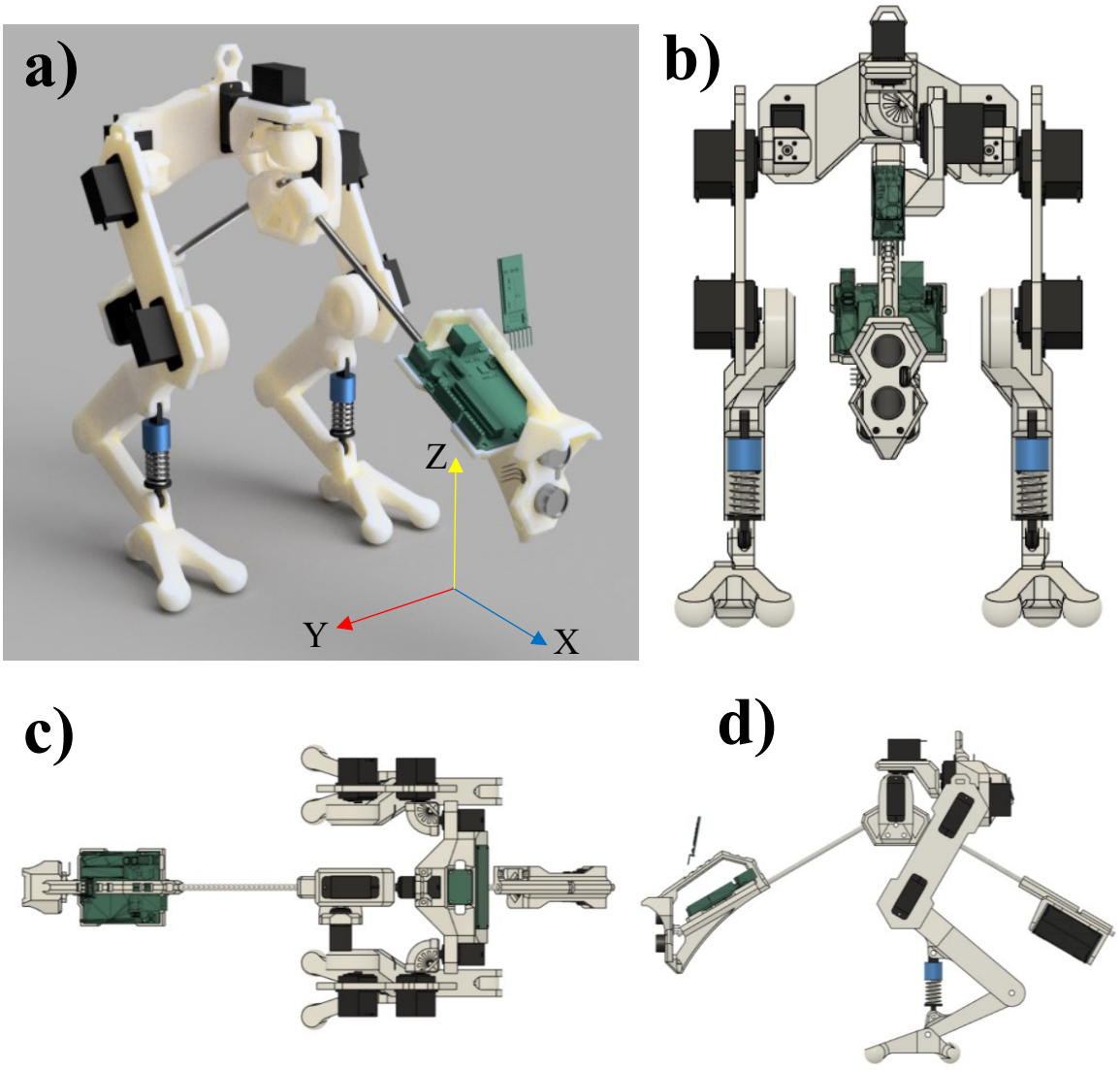


Figure 11: a) overall view of Pavo with the axis sign conventions utilised in the remaining report, b) Front view of Pavo, c) Top view of Pavo, d) Side view of Pavo

The robot was built around its components and was kept small to give the servos the best possible chance of functioning well. This resulted in a scale of 2.5:1 relative to the quail measurements utilised. Pavo is approximately 31cm tall, 21cm wide and 44cm long. The final physical product can be seen in Figure 12.

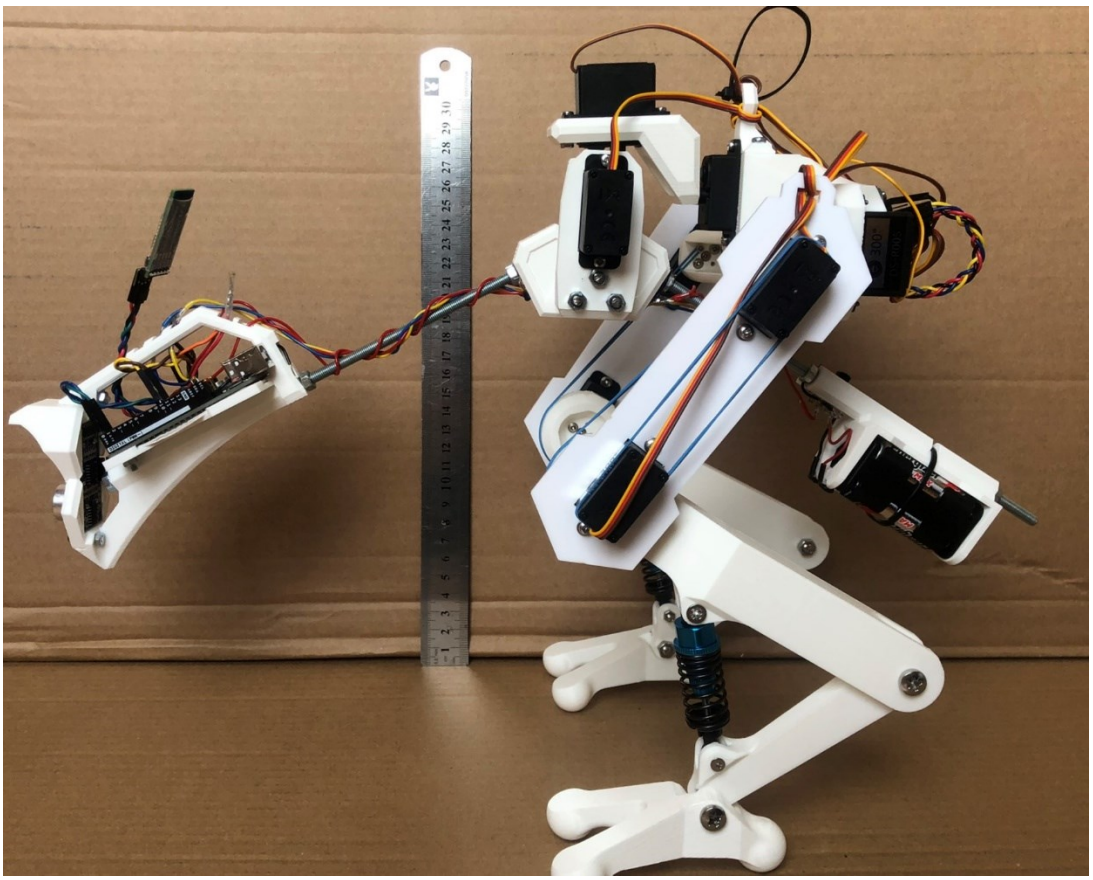


Figure 12: A side view of the finished robot

It was decided early on in the design process that 3D printing would be the main mode of manufacture, it was chosen for its relatively lower costs, quick turnaround, ability to create high complexity parts, and lower weight materials (Polylactic Acid, PLA plastic was utilised for its rigidity). These parts were fastened together with assorted nuts and bolts. Two notable non-3D printed structures include the thighs and spine. The thighs were laser cut from 3mm acrylic as they did not require the complex mounting points that other parts required 3D printing to achieve. The spine is comprised of two 23cm long 5mm threaded rods as these two sections were simple but load-bearing, likely requiring thick plastic that would have dramatically increased weight.

The components were assembled and electronics installed. A labelled circuit diagram of the electronics onboard can be seen in Figure 13.

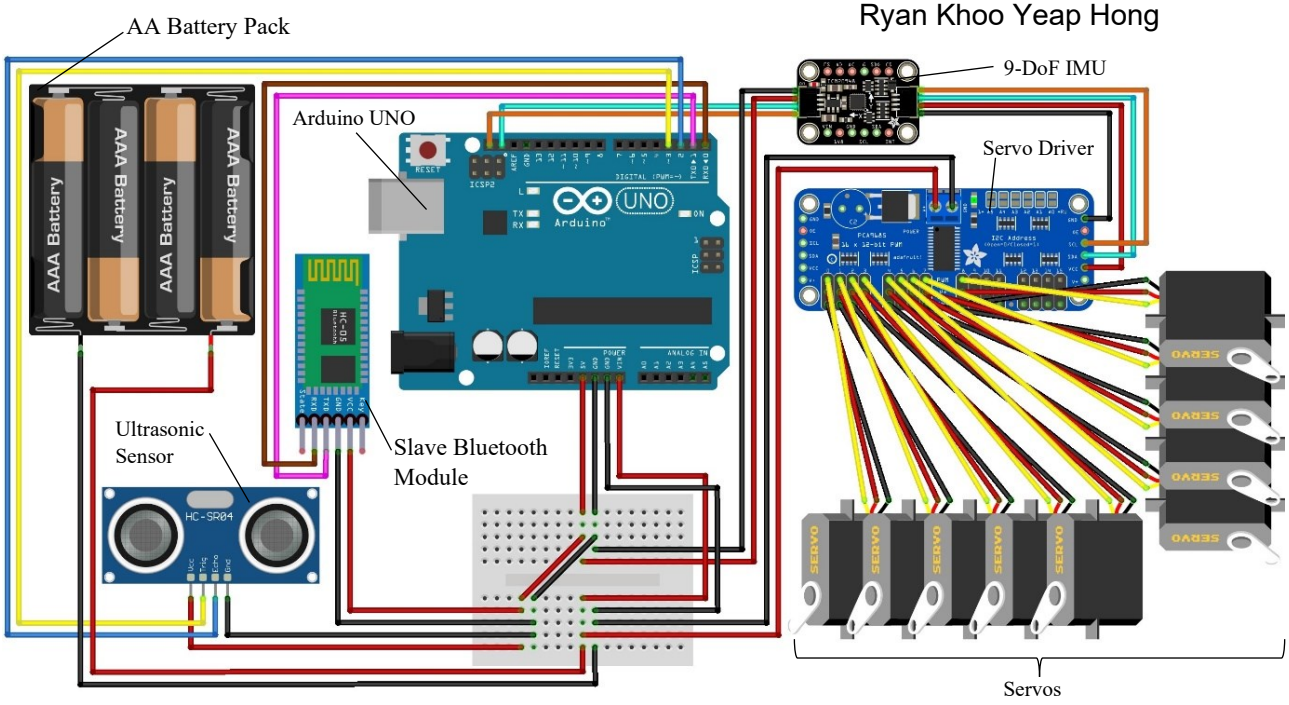


Figure 13: A labelled circuit diagram illustrating the connections between components onboard Pavo

Notable design features and their motivations are as follows:

Pavo's "head" can be seen in Figure 14 below. It is mounted to the 5mm threaded rod and is secured between two nuts. The head assembly is made of three 3D printed parts; the front plate housing to mount and protect the ultrasonic sensor from damage, the top rail to provide structural support to the assembly and provide guides for the wires, and the main plate to house the Arduino and attach the assembly onto the threaded rod.

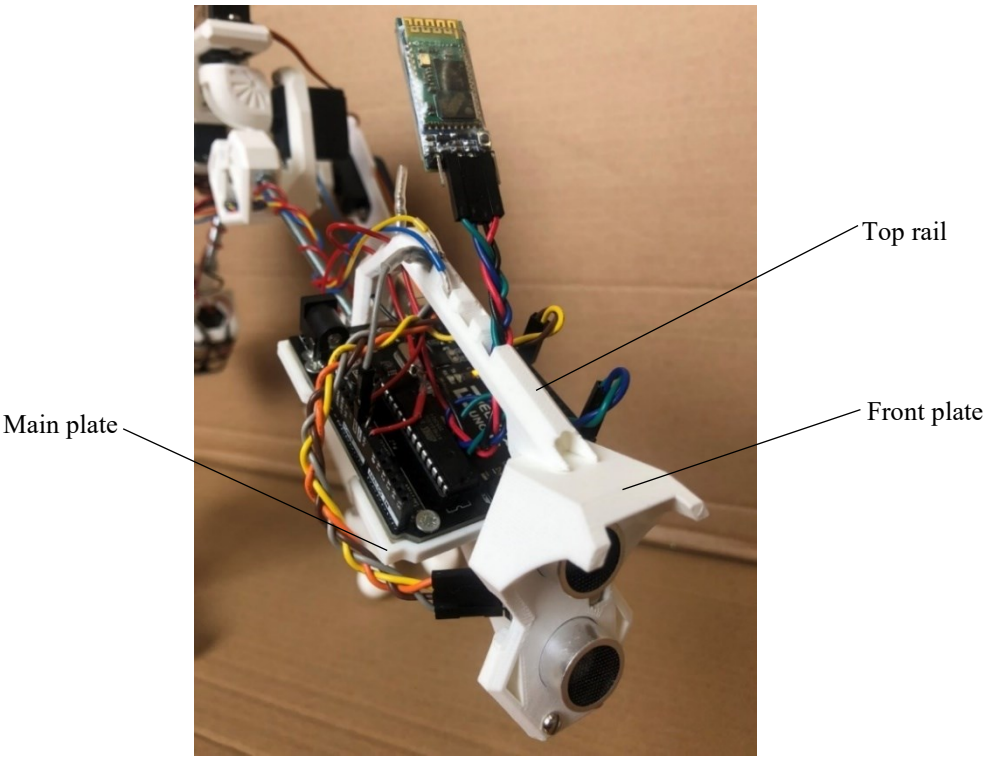


Figure 14: The "head" of Pavo

The hip assembly, representing the pelvis of the quail, houses three servos, the servo driver and the onboard IMU which are mounted securely to the 3D printed piece. As all the servo wires are routed to this area, a hole and cavity were designed into the piece to both save weight and provide an area for excess wires to be organised. A wire guide was also installed at the top of the hip assembly to secure the central servo and provide a guide for the wires from the three spine servos.

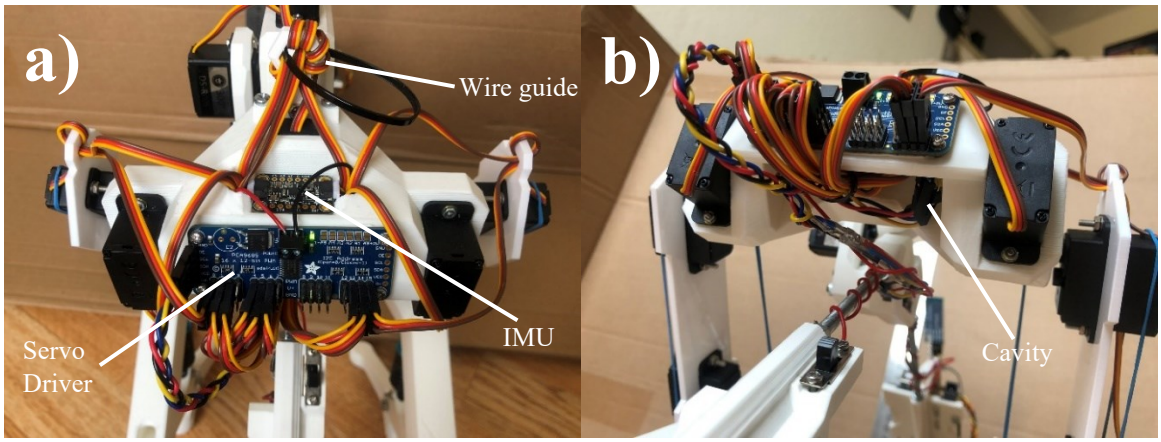


Figure 15: a) A top view of the hip, embedded IMU and servo driver b) A bottom view of the hip

The “tail” (Figure 16), similar to the “head”, is attached to the 5mm threaded rod and secured by two nuts. It was designed in a way that allows the tail to be adjusted up and down the rod as required to keep the CoM of the spine assembly below the hip. The tailpiece houses the battery pack and is secured onto the rod via a cable tie. A switch controls the power to the Arduino and other electronic components. It can also be noted that the electronic ground wire has been secured to the threaded rod itself, removing the need for a dedicated grounding wire throughout the robot, similar to a car chassis.

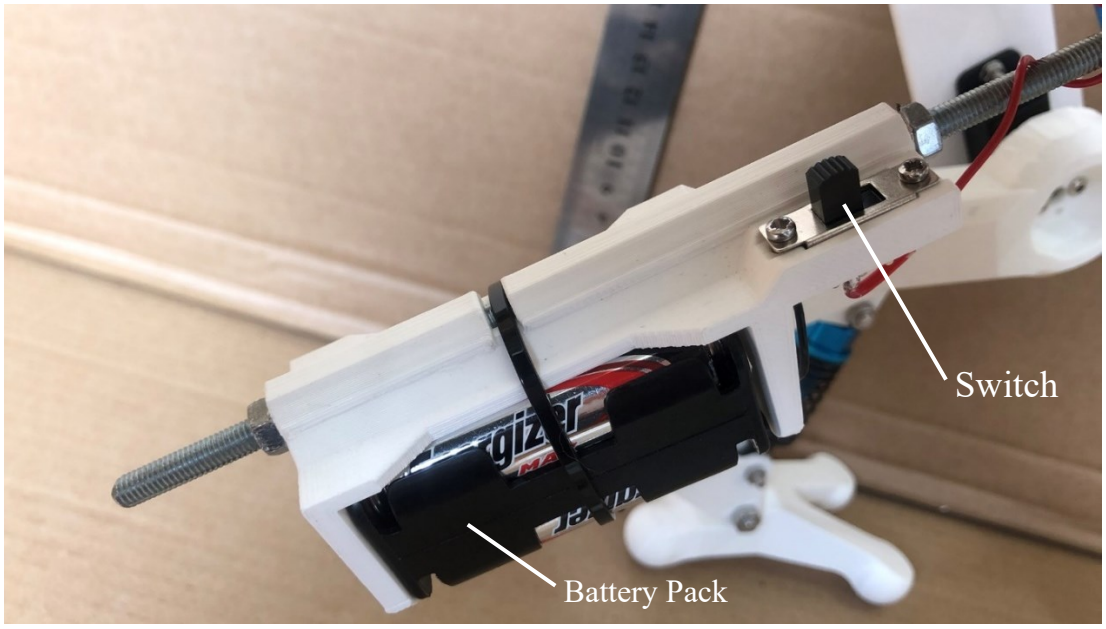


Figure 16: The “tail” of Pavo, housing the batteries and a power switch, both labelled

Pictured in Figure 17, the three servos suspending the spine can be seen along with the piece that connects the two threaded rods at a 120° angle. It has been designed to allow the passage of wires and be 3D printable while also being suitably robust as the two weighted rods produce large leveraging forces on the piece.

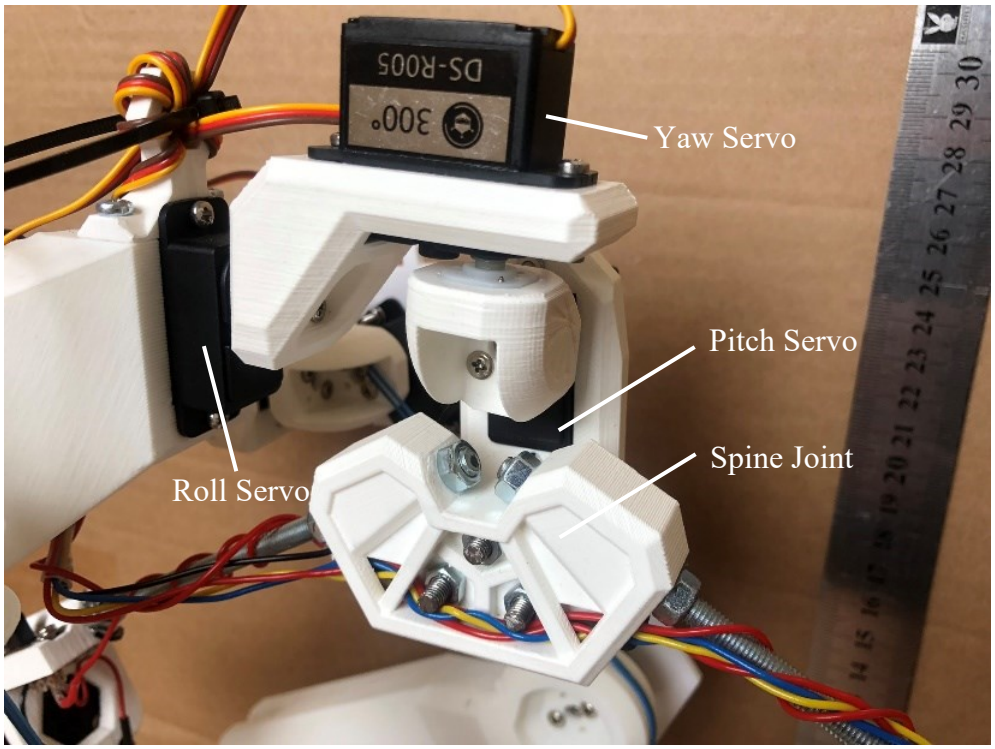


Figure 17: The middle section of Pavo with the spine joint and servos labelled

The implementation of the passive spring-damper can be seen in Figure 18, theoretically allowing the foot to flex upon contact with the ground and reducing bounce.

Figure 18: The left foot assembly of Pavo with spring damper labelled

Finally, the feet can be seen in Figure 19, designed to mimic the foot of a bird, it has a triangular footprint of 4cm by 7cm to allow Pavo to stand still without constant corrections.

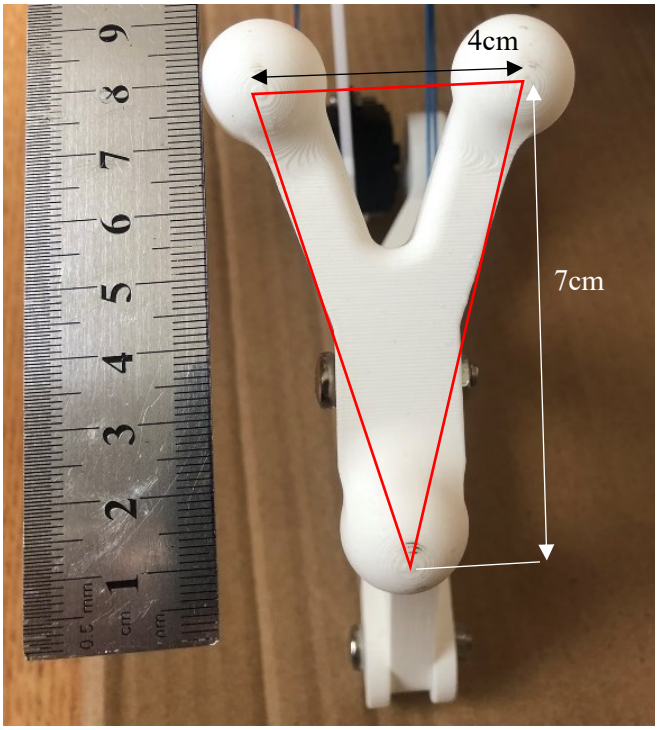


Figure 19: The foot of Pavo with the distances between the three contact points marked

A controller was also built to house the Arduino, Bluetooth module and batteries for the wireless control of Pavo (Figure 20, an engineering assembly drawing may be found in the appendix, A.9). A switch toggles power to the board and Light-emitting diodes (LEDs) indicate power and controller inputs. Two spools have been designed into the side of the controller to allow the wire of the controller to be neatly coiled when not in use. A circuit diagram detailing the connections within the controller can be seen in Figure 21.

The manufacture of Pavo involved the use of university workshops/facilities, safety guidelines were adhered to at all times, further elaborated on in the risk assessments found in the appendix (A.4).

Figure 20: The finished controller with remote

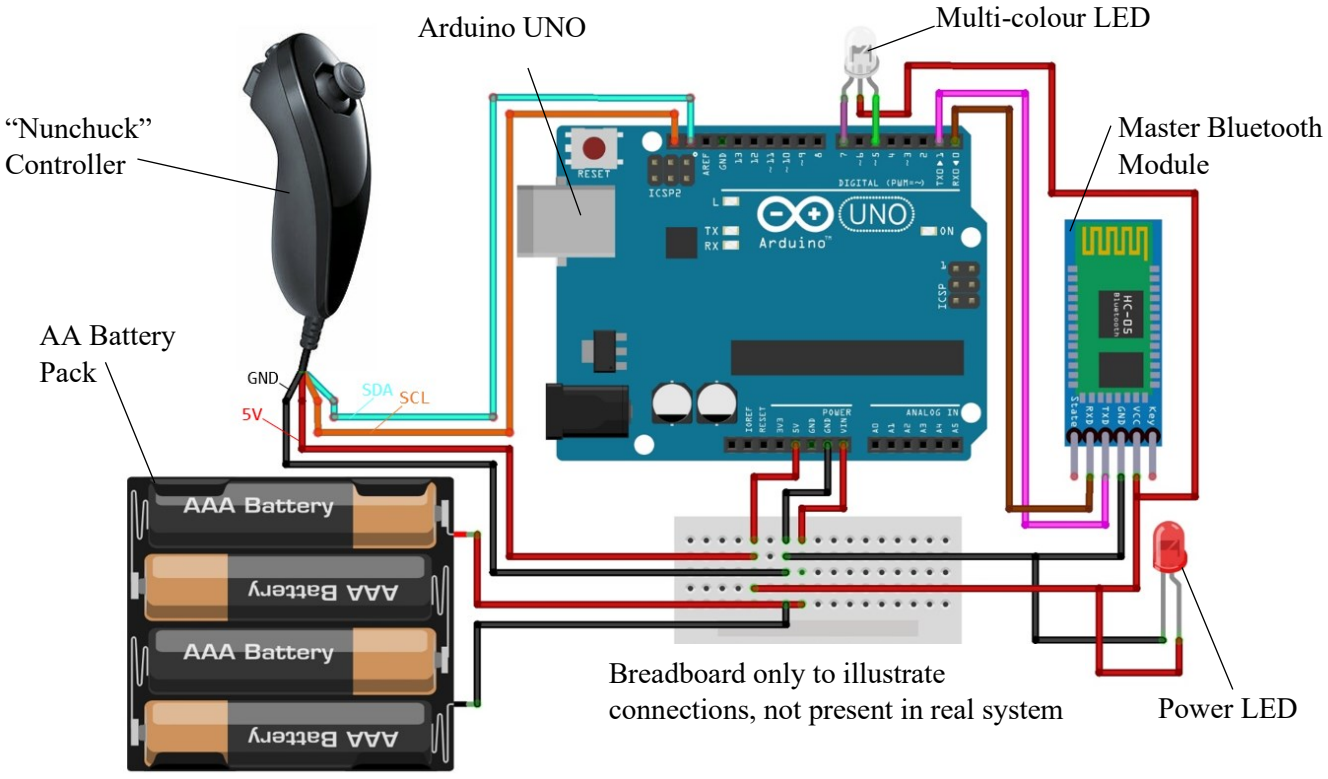


Figure 21: A labelled circuit diagram of the electronics in the controller

3. Control Theory and Programming

3.1. General Approach and Overview

Bipedal organisms found in nature exhibit incredibly complex and sophisticated behaviours with respect to governing the appendages concerned with their bipedal motion. They are able to take into account innumerable environmental variables and utilise them to best traverse their terrain in a desired direction/fashion. These systems may be referred to as central pattern generators (CPGs) (Ijspeert 2008).

The objective of the control system is then to imitate a CPG found in nature that is capable of accepting non-rhythmic, external data and produce an output in the form of rhythmic coordinated footsteps that dynamically adapt and adjust based on those external sensory inputs.

A secondary but important goal is to ensure the system (and code) is structured in a future-proofed manner that allows upgradability. It is for this reason that the code (available in the appendix, A.8) is broken down into many individual functions that interact with each other in a chain, each receiving data, processing/using it, and passing new data to the next function. This individual code structure allows further complexity/sophistication to be implemented in various aspects of the program without interfering with the functions of the rest of the code. It is however not as efficient as a more cohesive system that works in unison to produce the same effect faster and with fewer resources, this is further discussed in section 4.3.3.

In essence, the system designed is governed by two principal variables; modifier variables “xMod” and “yMod”, both ranging from -100 to 100 that motivate movement forwards/backwards and right/left respectively. These values are dynamically tuned to allow Pavo to balance and walk.

The system can be viewed in two parts: The first encapsulates all functions involved in obtaining these two variables at any one point in time, derived from the external inputs (the IMU data, obstacle sensing and controller input). The second involves the generation and execution of a cyclic gait pattern based on these variables. A high-level overview of the control system can be seen in Figure 22 below.

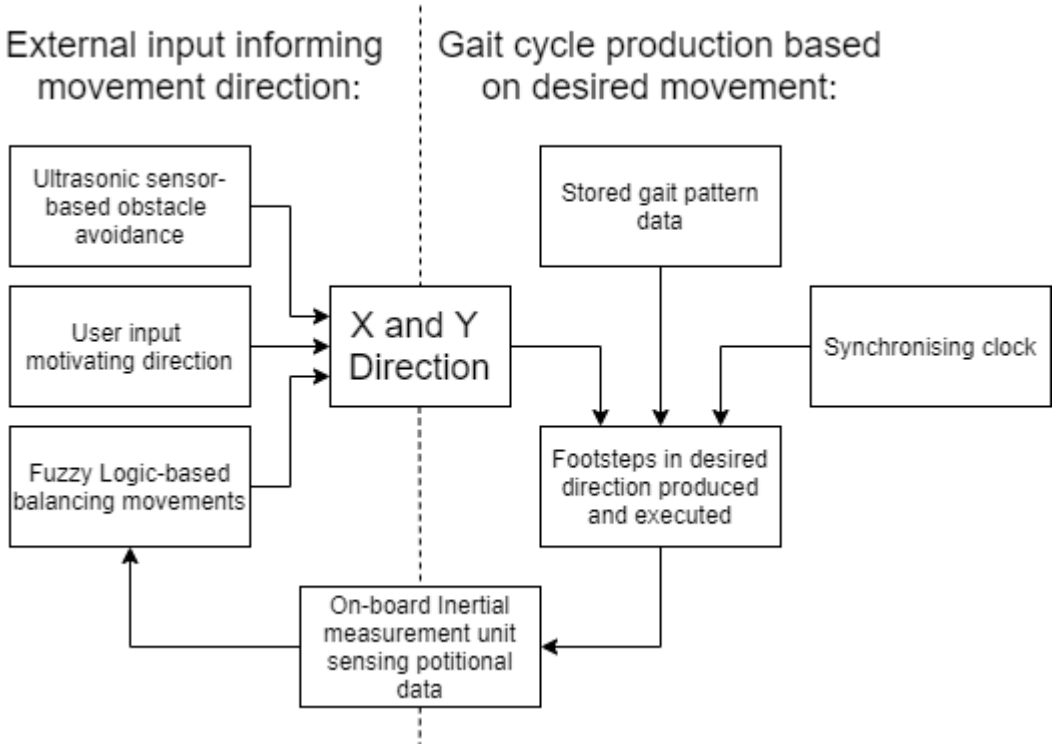


Figure 22: A high-level diagram illustrating the flow and functions of the control system designed

At over 800 lines (including comments and spaces to enhance readability), the code has been broken down into numbered sections for efficient navigation, it is punctuated by long lines of easily identifiable numbers (see code in the appendix, A.8.2). This structure also allows for all tuneable variables (such as gait speed and step size) to be placed at the top of the code to allow for quick intuitive numerical adjustments in a “control panel” like fashion.

3.2. Controller Code Theory/Bluetooth Communication

Utilising the Bluetooth module specified in section 2.2, the modules were first configured and paired as a “master” (the controller) and “slave” (onboard Pavo). Once paired, the master circuit is able to send data wirelessly to the slave circuit.

As the controller only sends simple directional information, a simple numbered command was utilised to send data in lieu of a more traditional data packet approach. This was done to reduce the code required on the onboard Arduino to save memory and processing power.

When input from the “nunchuck” controller is detected, the Bluetooth module transmits a numerical variable containing the command and its magnitude. A table of the commands implemented can be seen in Table 1 below.

Input	Command State	Sub-States (magnitude)	Description

Joystick North	10	10-19	Walk forward
Joystick East	30	30-39	Stride right
Joystick South	50	50-59	Walk Backwards
Joystick West	70	70-79	Stride Left
C Button + Joystick Pitch	110	110-119	Spine Pitch
C Button + Joystick Roll	120	120-129	Spine roll
C Button + Joystick East and West	130	130-139	Spine Look Left and right

Table 1: A table illustrating the possible commands that are sent over Bluetooth

For example, if the “nunchuck” controller sensed a maximum forward input, it would send a command of 19 to Pavo. The first number “1” indicates a command to move forward, and the second number “9” indicates that it should move forward at a maximum magnitude. On-board, the slave module would receive the command, note that it is a number between 10 and 19, categorise it as a “move forward” command, subtract 10 to obtain the magnitude of 9, and use this variable to obtain the direction modifiers (which are also based on IMU and obstacle sensor data), further discussed in section 3.4.7.

Many more command states are possible with the functionality of the “nunchuck” controller and a more complete table of initially planned commands can be found in the appendix (A.5).

LEDs were also utilised to provide visual feedback to the operator, with a green led indicating if the controller is turned on, and a multi-coloured LED turning green, red or orange depending on the command sent, these two LEDs can be seen in Figure 21 above.

3.3. Fuzzy Logic

Traditional methods of bipedal robot control require high volumes of complex dynamic calculations that were deemed unfeasible due to the limited speed, memory, and processing power of the Arduino UNO utilised in Pavo. As such, a fuzzy logic-based approach was used to off-load the computational power required from the Arduino onto a computer.

Another reason that fuzzy logic was chosen over more traditional methods is the lack of position feedback and complexity of Pavo, resulting in a mathematically ill-defined system. Traditional methods require a complete simulation of the robot to predict its future states. These methods require the simulated robot to be a highly accurate approximation of its real-world counterpart, requiring weight, motor positions, etc. On top of that, environmental data is also required, such as surface incline, uneven terrain information and surface

frictions. Therefore, generating a reliable and accurate model/representation of Pavo and its immediate environment for use in a traditional control system was deemed impractical.

Fuzzy logic-based systems, appropriately refined, are able to replace highly complex mathematical systems and has been utilised in this context to replace the balancing system required to prevent the bipedal robot from falling over.

The Fuzzy Logic Toolbox (V2.8) by MATLAB was utilised in the building of the system as it allows for smooth integration with the MATLAB coding environment required to produce a dataset that can be integrated into Arduino code, further specified in section 3.4.3.

The MATLAB toolbox allows both Sugeno and Mamdani style inference systems. The latter was chosen for its simpler, more intuitive fuzzy inference system, where human interpretable rules are used as opposed to Sugeno's less readable but more computationally efficient weighted average system. (Kaur and Kaur 2012).

Based on the cart-pole problem by Douglas (2021), a fuzzy logic system mimicking the natural response of accelerating in the direction of a fall was utilised to implement a dynamic balancing system in Pavo. To do this, the IMU data, namely the angular position and velocity were utilised as inputs to produce a directional output. Both values are first mapped to an integer between -100 and 100 based on experimentally obtained expected maximum/minimum values. This allows the fuzzy logic inputs to be adjusted in the code, reducing the need to regenerate a lookup table in MATLAB for every minor iteration. This was also done to avoid the use of decimals that take up much more memory than whole numbers in the Arduino code.

These normalised inputs are then fuzzified into two fuzzy sets each, representing the degree of negative or positive angle/angular velocity as seen in Figures 23 and 24 below.

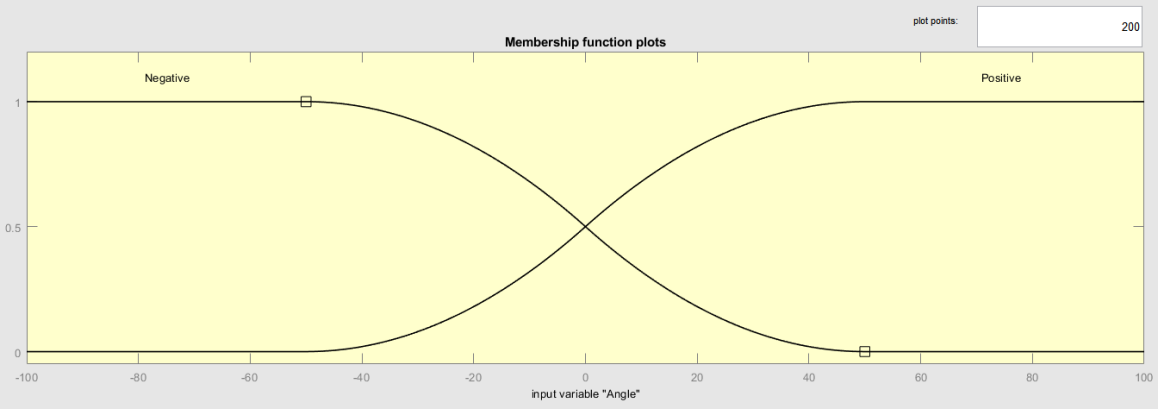


Figure 23: Membership functions of crisp angle input

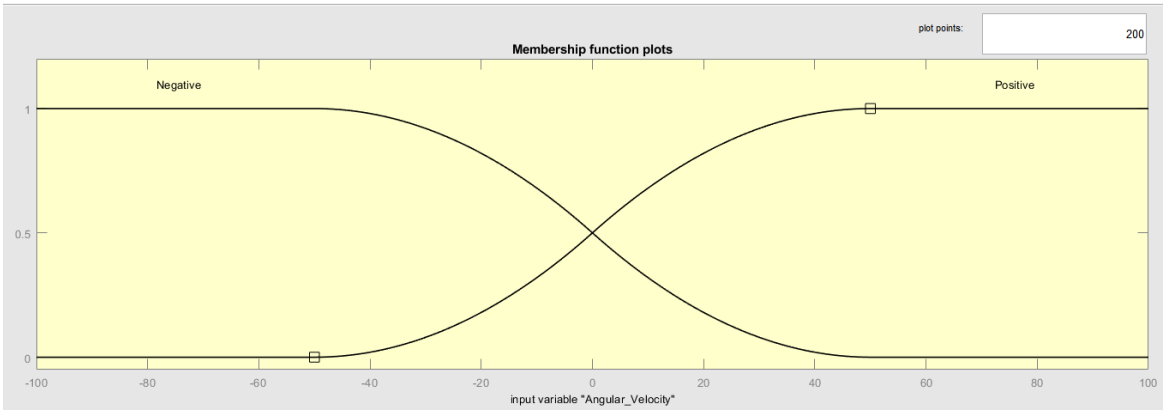


Figure 24: Membership functions of crisp angular velocity input

These four fuzzy variables are then processed with the following inference rules into four new fuzzy output variables:

1. If (Angle is Negative) then (Speed_& Direction is Walk_Negative)
2. If (Angle is Positive) then (Speed_&_Direction is Walk_Positive)
3. If (Angular_Velocity is Positive) then (Speed_&_Direction is Run_Positive)
4. If (Angular_Velocity is Negative) then (Speed_&_Direction is Run_Negative)

This is based on the natural reaction of walking in the direction of a fall to prevent it, with the speed of movement dependent on the severity of the fall. These four outputs are then utilised in the output membership function plot in Figure 25 and crisp numerical outputs obtained via the centroid method.

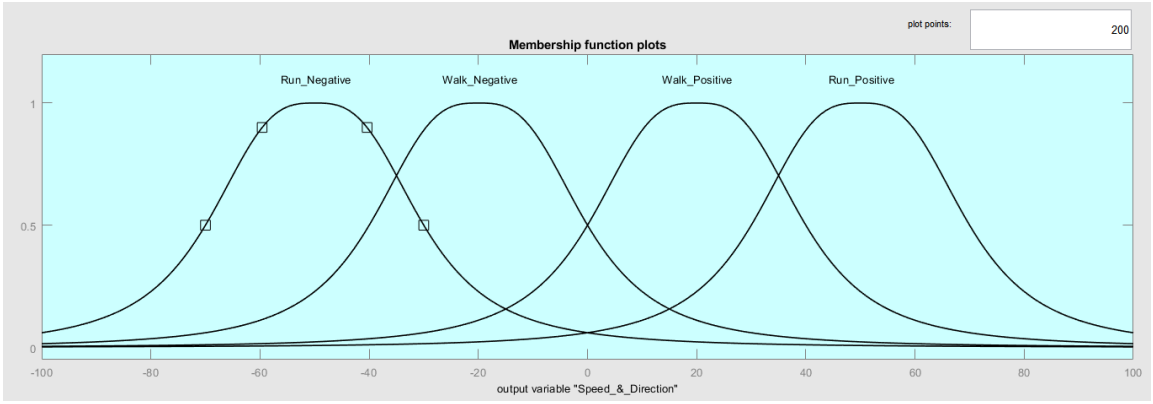


Figure 25: Output membership function plots utilising four fuzzy output variables

An example of this can be seen in Figure 26.

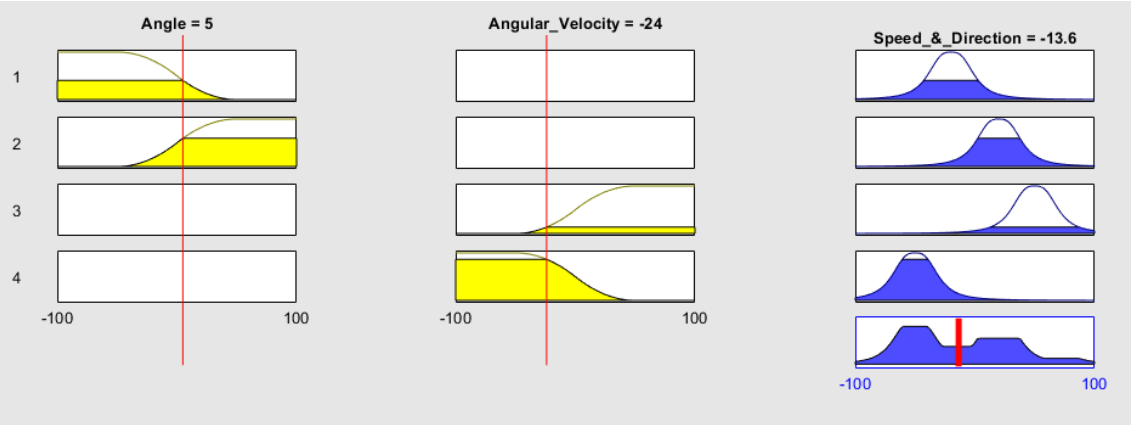


Figure 26: An example of the fuzzy logic system in use with normalised inputs (Angle = 5, Angular velocity = -24), and resulting normalised crisp output of -13.6

All possible input combinations were calculated and a surface plot generated, as seen in Figure 27 below.

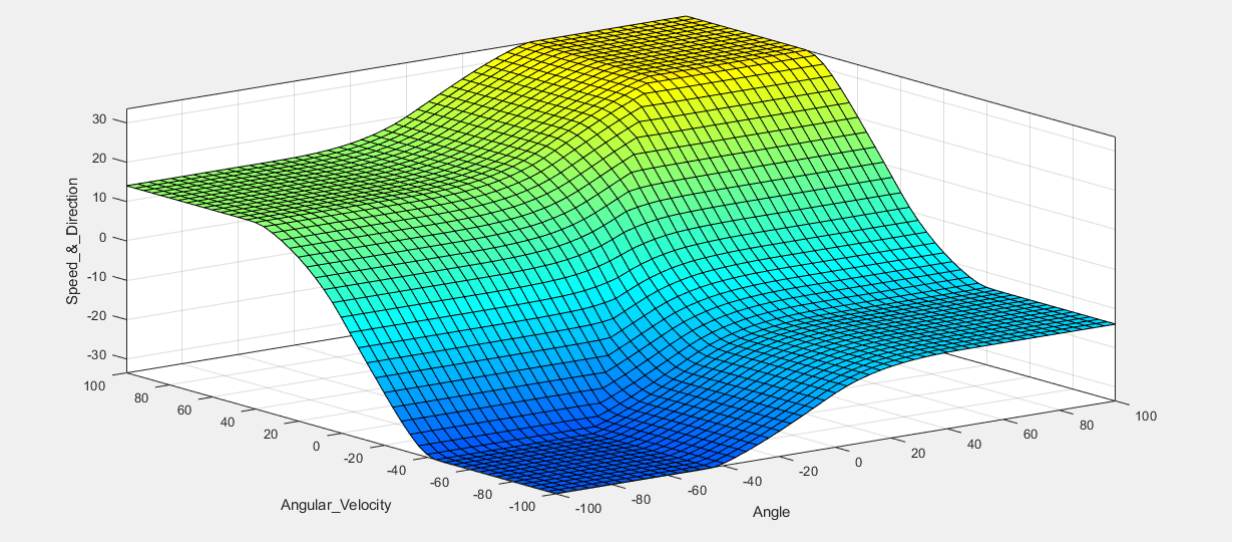


Figure 27: The resulting fuzzy logic surface plot with crisp outputs on the vertical Z-axis and crisp inputs of angle and angular velocity on the X and Y-axis

3.4. Code Details

3.4.1. Ultrasonic Sensor Implementation

The HC-SR04 ultrasonic sensor measures distance by producing an ultrasonic chirp and measuring the time taken for that chirp to be reflected. It calculates this via equation 1 (accounting for the distance travelled there and back by multiplying by 0.5):

$$Distance\ (m) = Duration\ (s) * 340(m/s) * 0.5 \qquad Eq.\ (1)$$

This was implemented without utilising a delay function that would interfere with the rest of the code. A method utilising the “micros()” function was used to accomplish this and can be seen in the code found in the appendix (A.8.2).

This distance check is executed once every footstep cycle to reduce the computational load required, this still amounts to multiple measurements a second which is adequate for the prevention of collisions with objects.

3.4.2. IMU Implementation

The IMU was utilised to obtain the angular position and angular velocity of Pavo's hip (where the IMU is embedded). The IMU provides raw data in the form of acceleration from the accelerometer and angular velocity from its gyroscope. Both were combined to produce the best possible approximation of its angular position.

The accelerometer was first utilised to obtain the pitch and roll of the robot utilising equations 2 and 3 below, calculating the resultant vector angle relative to gravity, converting it into degrees and correcting an experimentally obtained error value:

$$XAngle = \left(atan\left(\frac{Y}{\sqrt{X^2}}\right) + Z^2\right) \times \frac{180}{\pi} + error \qquad \textit{Eq. (2)}$$

$$YAngle = \left(atan\left(\frac{-X}{\sqrt{Y^2}}\right) + Z^2\right) \times \frac{180}{\pi} + error \qquad \textit{Eq. (3)}$$

The Gyro angular velocities are then manually integrated with equation 4 below to obtain an alternative set of pitch and roll values.

$$New\ Gyro\ angle = Previous\ Gyro\ angle + Angular\ velocity * Time\ since\ previous\ gyro\ angle \qquad \textit{Eq. (4)}$$

These two sets of angles derived from accelerometer data and gyro data were then combined to obtain the best approximation of the angular positioning of Pavo.

The angular velocity about the X and Y-axis is also utilised by the fuzzy logic system. They are provided by the gyros with no extra calculations necessary.

3.4.3. Fuzzy Logic Implementation

The pre-calculated fuzzy logic data of Figure 27 above is integrated into the Arduino code by translating the surface into a 2-dimensional look-up table, similar to the work of Sobhan et al. (2009), where inputs of angle and angular velocity as row and column indexes return an appropriate directional output. This was accomplished with MATLAB code (see appendix A.6). The surface was discretised into a 51 by 51 matrix array, allowing the Arduino to store the table in its flash memory. The MATLAB code written allows for adjusting the size of the matrix, with larger tables providing better accuracy from smaller increments, allowing for future improvements should a processor with more memory be utilised.

The raw IMU data was smoothed with an exponential filter to reduce the noise from the vibrations of the physical robot. This was employed utilising equation 5:

$$x_{Filtered} = W \times x_{New} + (1 - W) \times x_{Old} \qquad \textit{Eq. (5)}$$

With W being a tuneable weight factor to modify the nature of the filter, $x_{Filtered}$ being the filtered value, x_{New} being the new input value and x_{Old} being the previous filtered value. The results of the filter can be seen in Figures 28 and 29.

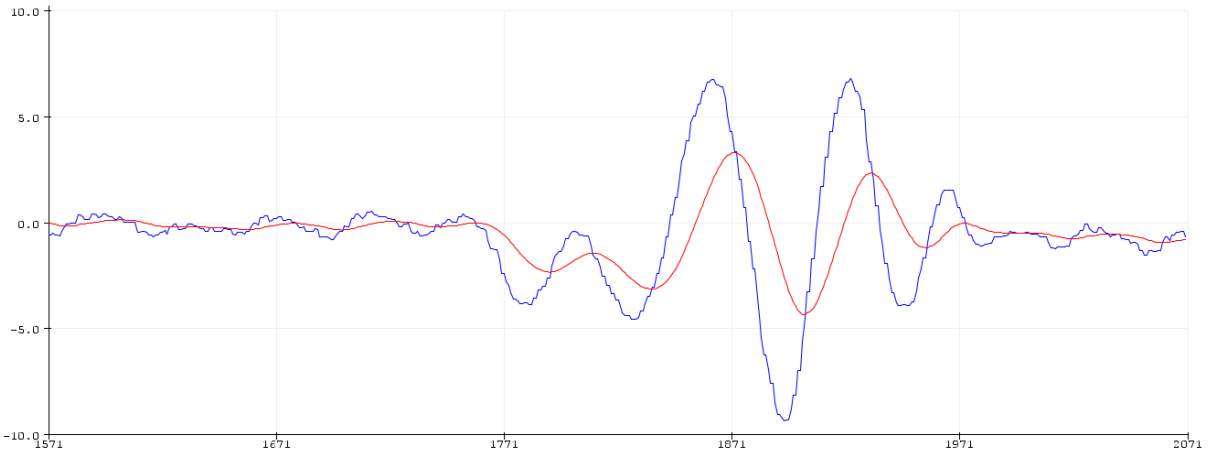


Figure 28: The raw roll value (blue) and the filtered, smoothed roll value (red)

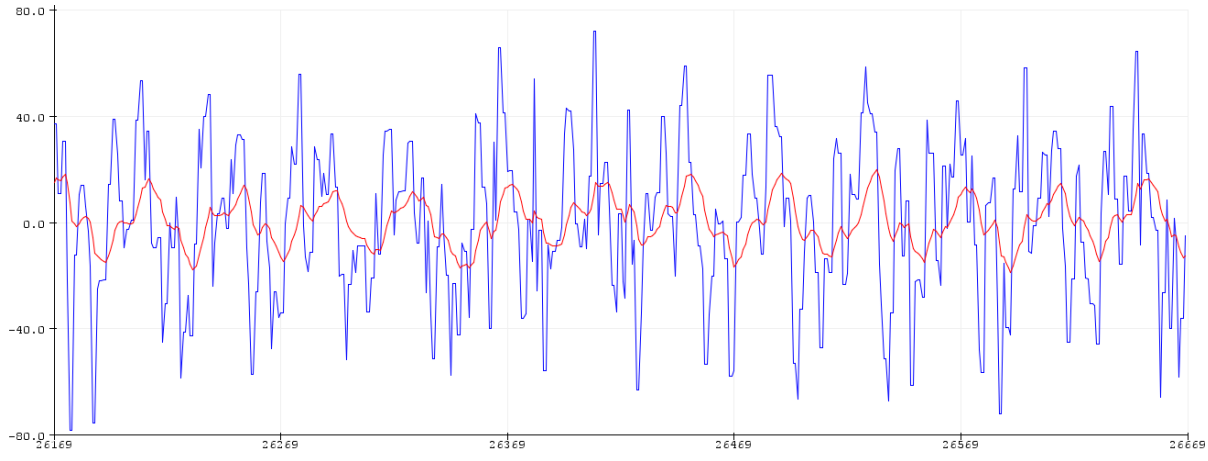


Figure 29: The raw Y-axis gyro value (blue) and the filtered value (red), with the filter effectively extracting a steady rhythm out of a noisy input resulting from the gait cycle of Pavo

3.4.4. Servo Set Structure

Data for any given position is stored in an array of ten variables, with the first nine values corresponding to each of the nine servos on board, and the tenth value utilised by the "executePosition()" function to determine the number of transitionary states to generate

between the old and new position (detailed in section 3.4.8), an example of this structure can be seen in Figure 30 below.

Figure 30: The structure of a position set utilised in the code, with nine servo values and a tenth "divisions" value that determines the number of transitionary values to be generated

These values correspond to the positioning of each servo and have been normalised to a value between -100 and 100, representing a servo's minimum and maximum values respectively. This allows for a custom range of movement to be set for each servo to prevent them from moving to positions beyond their range, damaging components and to remove the need to recall the specific ranges and values of each servo.

This system also utilises a pre-set neutral position value, corresponding to the normalised "0" value. This allows Pavo to intuitively come to a neutral standing position when a set of only zeros are written to the servos, it also allows a convenient way of adjusting the neutral position of the servos with a single variable change. An example of this system is illustrated in Figure 31 below.

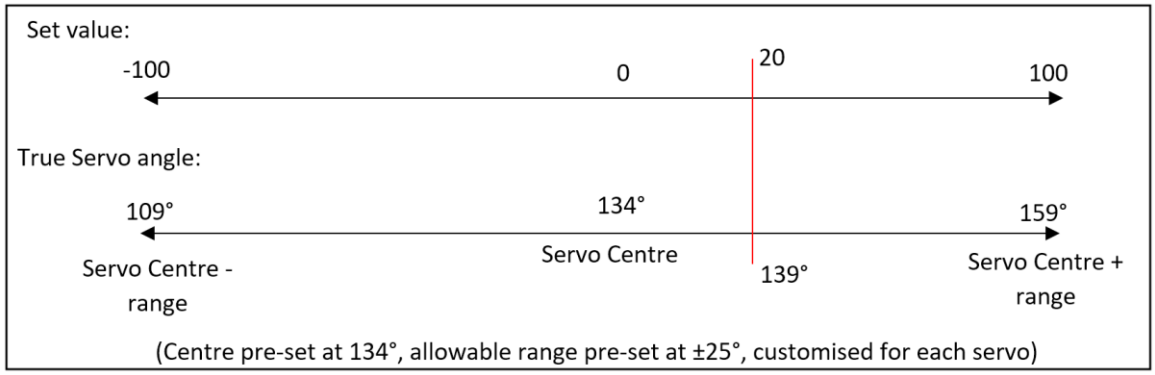


Figure 31: Parallel number lines illustrating the mapping of true servo angles to their normalised values utilised in the position sets with an example of a set value of 20 resulting in a servo angle of 139°

3.4.5. Cycle Stages

From the quail walk cycle (Figure 6, section 1.1), a gait pattern was generated by sectioning the gait cycle into eight distinct positions. With regards to a single leg, six positions correspond to ground contact, and two represent the leg while airborne and in the swing phase. Dividing a cycle into eight allows for the two legs to work in tandem (with an offset of four stages) to reproduce the walking cycle of a quail, with both legs in contact

with the ground (double support) for a total of 25% of each cycle, a good approximation of the experimentally obtained 26% by Abourachid et al. (2011). This is illustrated in table 2 below.

	Stage 1-2	Stage 2-3	Stage 3-4	Stage 4-5	Stage 5-6	Stage 6-7	Stage 7-8	Stage 8-1
Left Leg	Grounded	Grounded	Grounded	Air	Air	Air	Grounded	Grounded
Right Leg	Air	Air	Grounded	Grounded	Grounded	Grounded	Grounded	Air
Resulting Support	SS	SS	DS	SS	SS	SS	DS	SS

Table 2: A table showing the contact of each foot with the ground during the transitions between the eight states, and the resulting single support (SS) or double support (DS) state of a stage, with ground contact states indicated in grey

These eight sets were then each broken into three different aspects, one “empty” set, that represents a “running on the spot” movement, a “front/back” set, and a “left/right” set that when combined with an empty set would produce footsteps enabling front/back and left/right movement respectively.

These sets are detailed in table 3, and the resulting cycle of all eight “empty” and “front/back” sets can be seen in Figure 32.

Footstep stage:	% of cycle	Empty set	Front/back set	Left/right set
1	12.5%	Right leg up	L C3, R air	L C3, R air
2	12.5%	Right leg up	L C4, R air	L C4, R air
3	12.5%	Neutral	L C5, R C1	L C5, R C1
4	12.5%	Neutral	L C6, R C2	L C6, R C2
5	12.5%	Left leg up	L air, R C3	L air, R C3
6	12.5%	Left leg up	L air, R C4	L air, R C4
7	12.5%	Neutral	L C1, R C5	L C1, R C5
8	12.5%	Neutral	L C2, R C6	L C2, R C6

Table 3: A table describing the three aspects of each of the eight stages, with “L” denoting the left leg and “R” denoting the right leg, C1 represents the first contact of the foot with the ground, C2, the second, and so on, until the leg is lifted during the swing phase, indicated by “air”

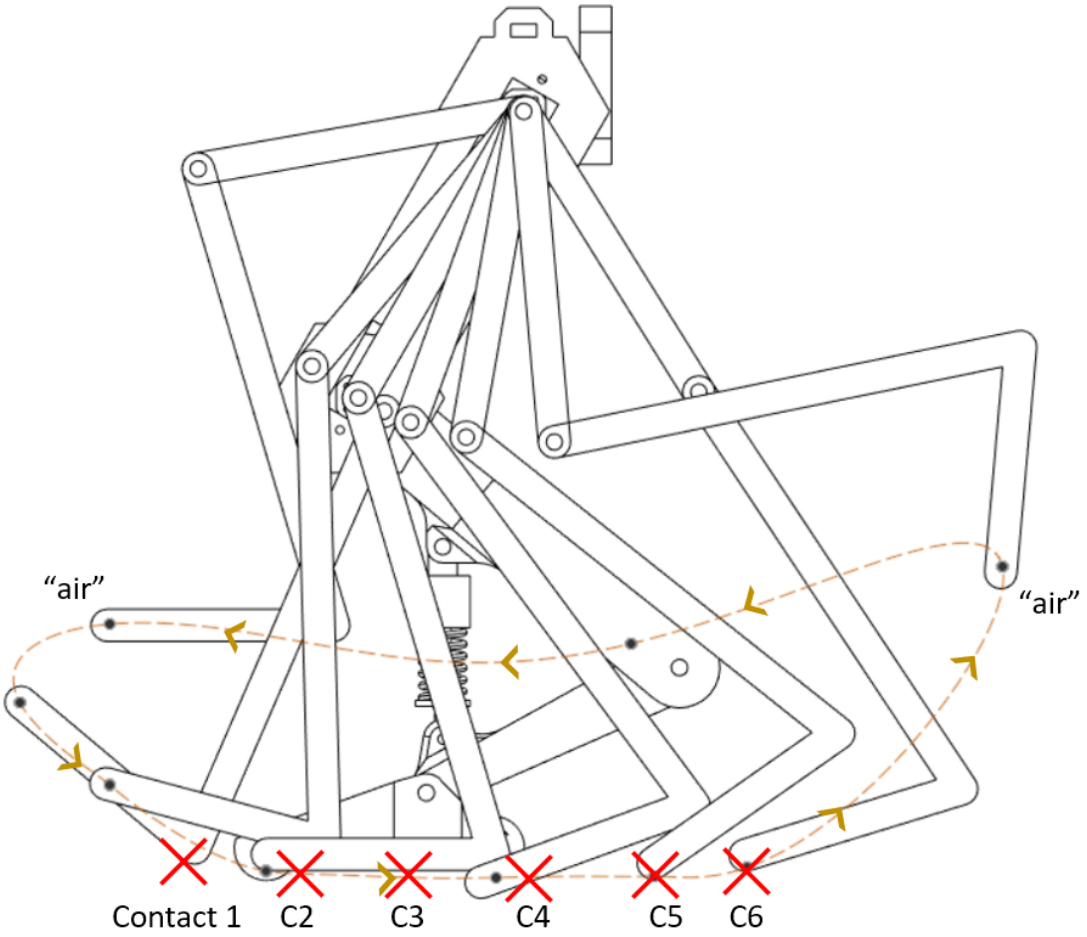


Figure 32: The resulting step pattern of eight “empty” sets combined with their eight corresponding “front/back” sets, with the ground contacts indicated by a red “X”

These three sets can be combined with different weights to produce footsteps in any front/back and left/right direction, determined by the “xMod” and “yMod”, detailed in section 3.4.7 below.

3.4.6. Cycle Stage Control

An independent system is required to coordinate these stages and to ensure the correct sequence of movements is executed. This is done by a cyclic global variable that denotes the current stage of the gait pattern (1 to 8). Upon completion of a stage, this “cycle stage” variable is incremented to the next, cycling through the stages. The code utilises this variable to call upon the appropriate stored movement sets to produce the next movement in the gait cycle.

A 9th stage was also created to represent “at rest”. This is to allow Pavo to come to a complete stop when the system is in balance. On start-up, Pavo remains in stage 9, and will only break into the cyclic stages 1 to 8 if the “xMod” or “yMod” surpasses a certain pre-set threshold, indicating a desire to begin movement. It is then possible for the system to break out of the cyclic gait back to the “resting” stage if both direction modifiers fall below

the threshold at any time during the cycle (indicating a balanced state), preventing unnecessary power consumption and component wear from “running on the spot”. This cycle is illustrated in Figure 33.

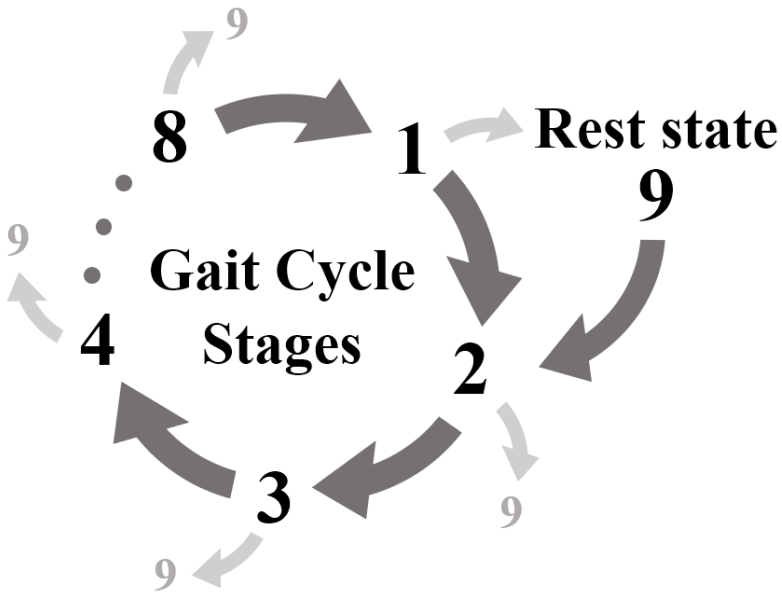


Figure 33: A diagrammatic representation of the cyclic stage variable system

3.4.7. Direction Defining and New Position Generation

To obtain the X-axis direction modifier “xMod”, the code first utilises the filtered angle and angular velocity about the Y-axis to extract an appropriate direction value from the fuzzy logic lookup table, it then takes into account the current command state from the controller (if any) and adds that value to the modifier, finally, it checks if the ultrasonic sensor detects an object closer than a predefined amount, and if it does, adds an appropriate negative value to the modifier to encourage deceleration or acceleration backwards. This system enables Pavo to avoid collisions without overwriting the fuzzy logic variables or operator inputs, allowing all three to operate together.

The Y-axis modifier “yMod” is similarly obtained by utilising the angular position and velocity about the X-axis to obtain a fuzzy logic output from the same lookup table and is modified based on input from the controller.

These two values (both between -100 and 100), along with the current cycle stage variable are used to extract the three appropriate sets for the current cycle stage and are combined with weights based on the modifiers to generate the next position set in the direction required. A graph illustrating examples of resulting directions and magnitudes can be seen in Figure 34.

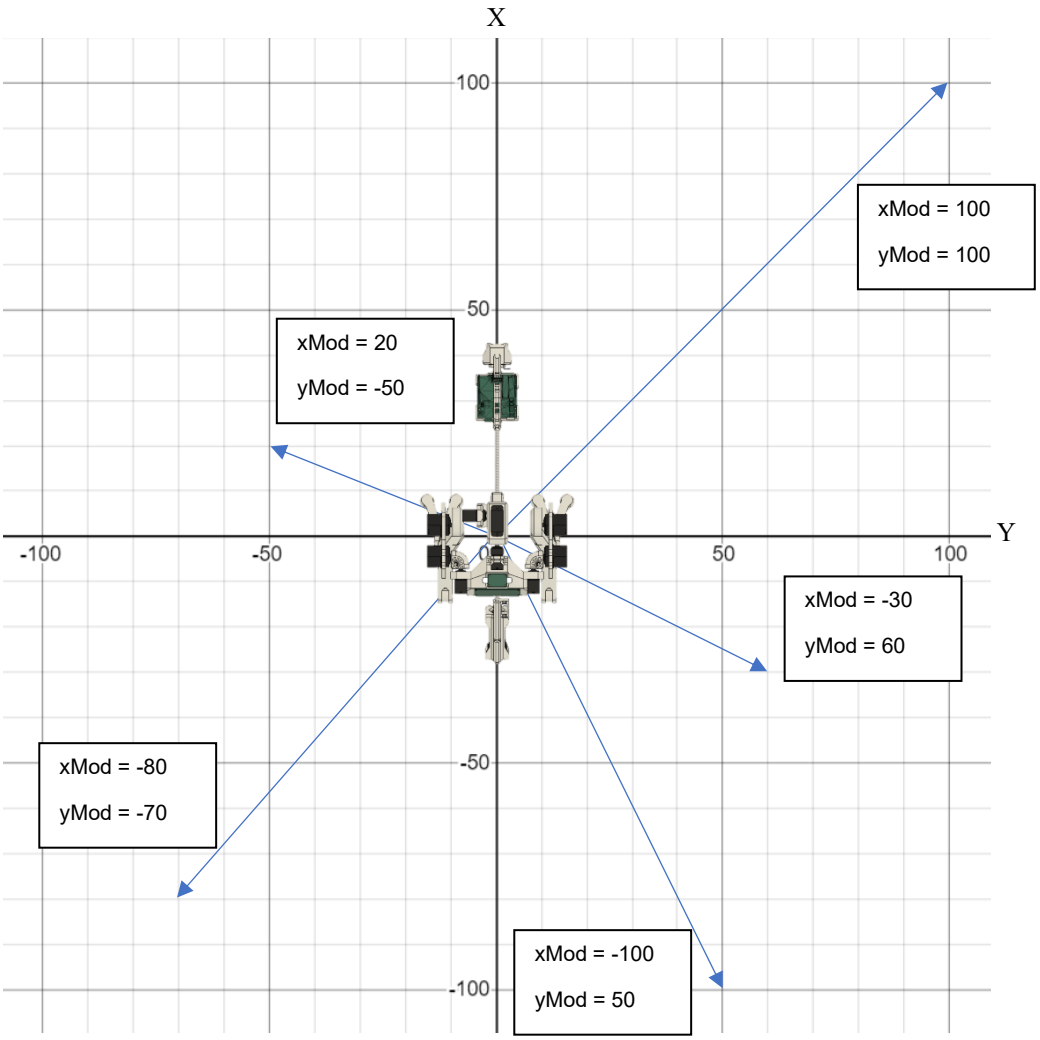


Figure 34: A graph illustrating various directions and magnitudes of movement derived from the X and Y modifiers, with the vertical X-axis representing walking forward/back relative to Pavo (pictured from above)

3.4.8. Eased Position Transitions

Once a new target position has been generated, Pavo's components must move to the new position in small increments to allow for speed control of the servos. This is done by noting the new and current position of a servo and populating the space between them with a number of transitionary positions based on the number of divisions required by the new set (as in Figure 30).

These transitionary states may be generated in three different styles as seen below in Figure 35; One linear transition for normal operation, and two sinusoidal transitions, one for beginning a movement from rest and the other slowing down the robot to a rest state.

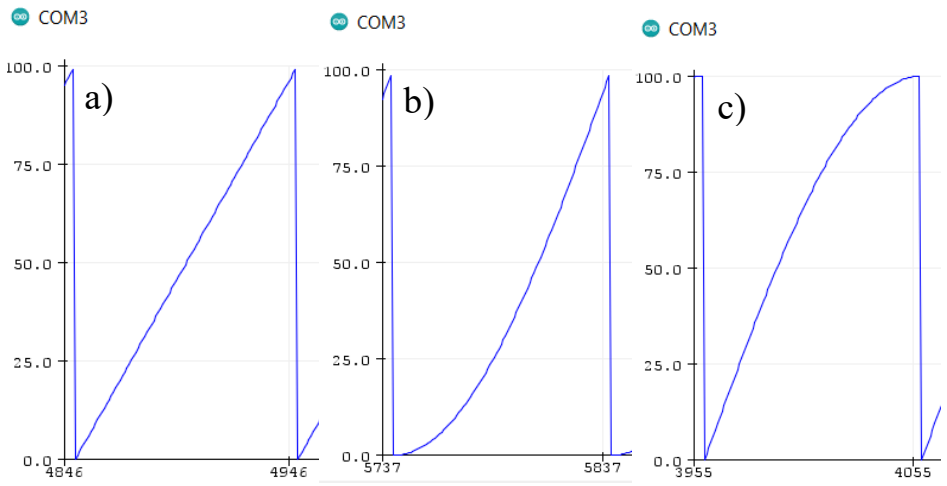


Figure 35: The three possible transitionary trends, a) Linear, b) Speeding up, c) Slowing down, with an x-axis of time and y-axis of servo position, 0 representing the old state and 100 the new target state

This is done by first producing an array of variables (on a scale of 0% to 100% as in Figure 35) with the appropriate transition style and length (length determined by the tenth "divisions" variable), then multiplying each element by the difference between the old and new value of each servo.

This multiplier array is produced in a "for loop" that loops according to the divisions required and generates the values one at a time until the full array has been populated, the equations 6 to 8 are utilised for each trend pattern.

Linear:

$$multiplier = \frac{1}{divisions} \times i \qquad \textit{Eq. (6)}$$

Slow down to rest:

$$multiplier = sin\left(\frac{\pi/2}{divisions} \times i\right) \qquad \textit{Eq. (7)}$$

Speed up from rest:

$$multiplier = cos\left(\left(\frac{\pi/2}{divisions} \times i\right) + \pi\right) \qquad \textit{Eq. (8)}$$

With i being the cumulative times the loop has run.

This transitionary array of servo positions is then placed into a "pending set container" (described below in section 3.4.9), and the process repeated for all remaining servos.

3.4.9. Pending Set Container and Periodic Position Execution

As each of the servo transitionary arrays is calculated, they are placed in a 2-dimensional array with ten columns to store the ten variables of a position set (see Figure 30), the number of rows is based on a pre-set variable indicating the maximum transition states allowed which may be increased with larger SRAMs (static random-access memory). This "container" stores the queued position sets that need to be written to the servos and is structured as in Figure 36 below.

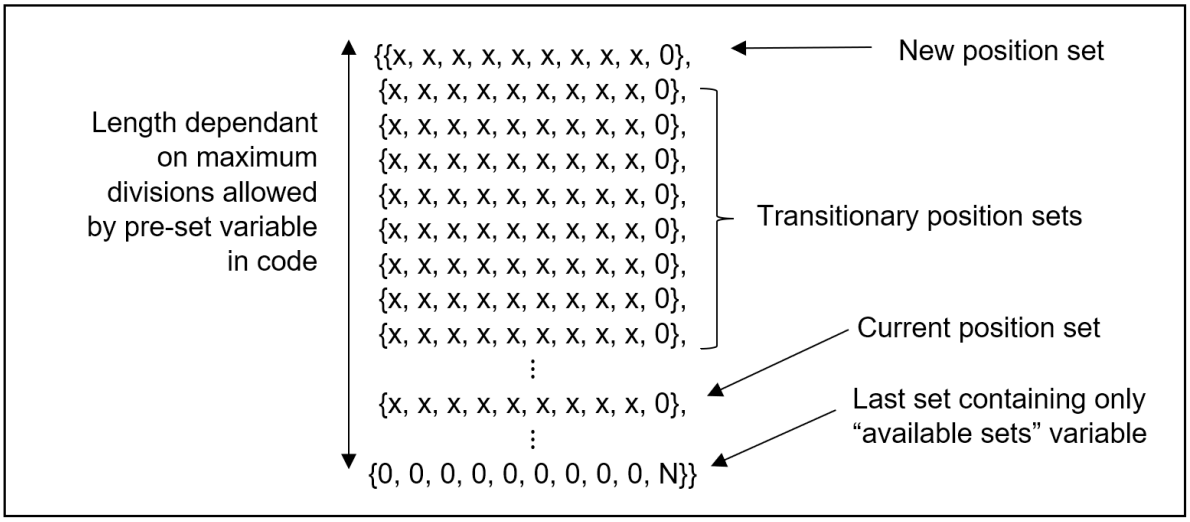


Figure 36: Format of "pending sets" container array, with "x" denoting a servo position value

Upon calculating a new transitionary array, the pending set is populated from the top down (with reference to Figure 36) and "N" updated with the number of rows added, this is the number of sets queued and available for execution.

The code periodically checks the pending set container (determined by a pre-set time-variable) for available sets to execute. If "N" is not 0, it will take the position set on the N^{th} row and execute those positions. It will then reduce N by one to allow for the next set to be executed on the next check. This repeats until N is 0 and Pavo has assumed the new target position on the first row of the container. The container is then open to be overwritten with the next set of transitionary position sets derived from a new target position and N reset accordingly.

All these systems work in unison to produce an upgradeable dynamic system that receives non-rhythmic external data and produces an adaptive rhythmic gait cycle; mimicking the central pattern generators found in nature. A more comprehensive version of figure 22 illustrating how the above functions are linked and function together can be seen in figure 37 below.

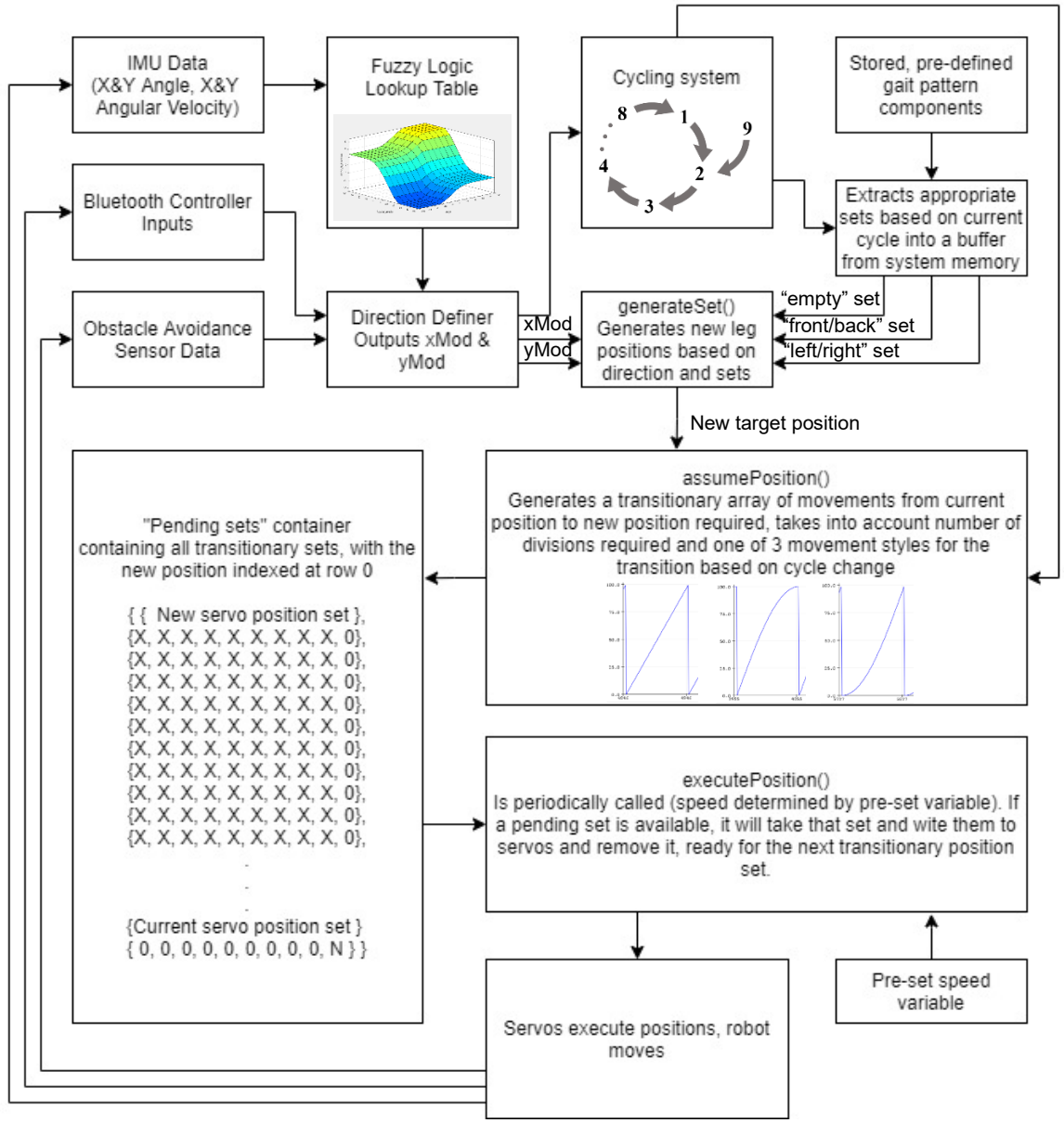


Figure 37 A control diagram detailing how all the individual functions detailed above in section 3 work together to imitate a CPG

4. Results and Discussion

4.1. Post-Assembly Adjustments

4.1.1. SRAM Limitations

Upon the initial implementation of the code above, it was found that the SRAM of the Arduino, where it stores variables, was overloaded. This resulted in the Arduino glitching and failing to run the code. It was concluded that the large arrays, namely the 51 by 51 fuzzy logic lookup table, pending set container, and the eight cycle stage positions, each broken into three sets ten variables long, were overutilizing the SRAM.

An Arduino library was utilised to store these large amounts of data in the much larger flash memory of the Arduino instead (via the PROGMEM function). This allows the SRAM to only store the data that is immediately required by extracting the data from flash memory into a buffer that can be overwritten once the data has been utilised (Andrews 2015).

Prior to this fix, the flash memory was only at 56% of its maximum capacity and SRAM was at its limit (72% of its maximum capacity to global variables, and the rest to local variables), crashing the program. Once implemented, the flash memory usage increased by only 6%, while the SRAM usage decreased by 29%, allowing the code to function as originally intended.

4.1.2. Servo Torque Limitations

Upon initial testing, it was found that the servos were vastly underpowered for the intended movements, with the servos unable to perform large movements without skipping steps and eventually failing due to the inherent moment applied to the knee motor due to the configuration of the leg.

A modification inspired by Badri-Spröwitz et al. (2022) was implemented to aid the knee servos in carrying their loads. The “Birdbot” developed by Badri-Spröwitz et al. utilises an avian-inspired leg clutching mechanism to achieve an energy-efficient gait, an important aspect of this are springs and pulleys in the leg that accept and store energy upon contact with the ground and re-release it when kicking off, imitating the ligaments and tendons found in nature as mentioned in section 1.1 (Badri-Spröwitz et al. 2022).

An analogous system was implemented by attaching a rubber band to the knee joints as seen in Figure 37 below.

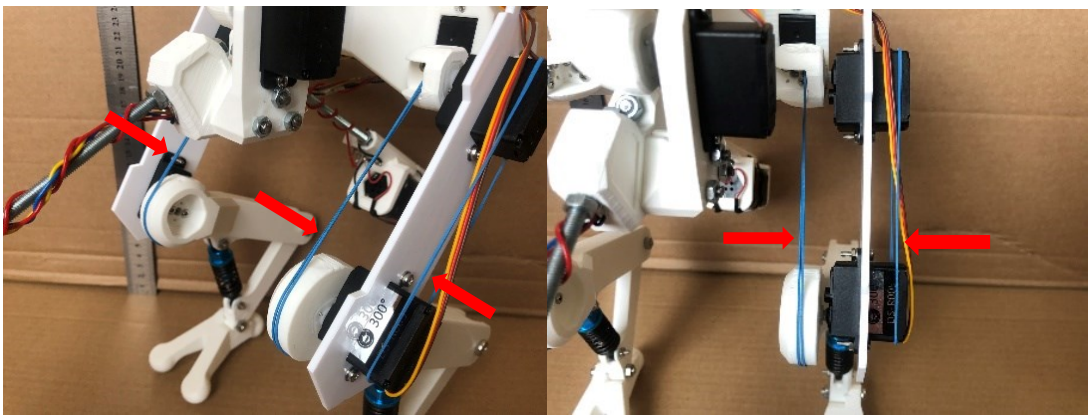


Figure 38: Two views of the added rubber bands (blue) attached to the knee to aid the servos, indicated by red arrows

A second rubber band was applied to the opposite side of the leg to counteract the flexing of the acrylic thigh piece due to the initial rubber band.

A diagram of the modification can be seen in Figure 38, where the tensioned band applies a force “A” on the knee joint, resulting in moment “B” about the pivot point, counteracting the inherent moment “C” resulting from its weight, reducing the resultant torque requirement on the knee servo joints.

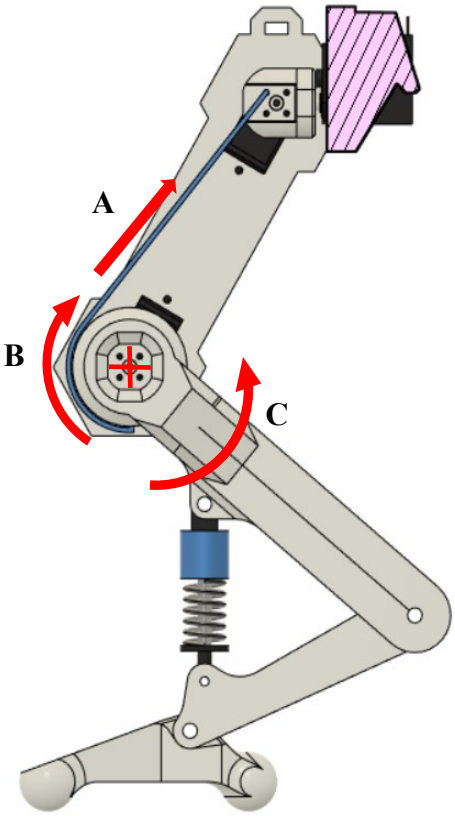


Figure 39: A force/moment diagram illustrating the rubber band modification (blue) counteracting weight induced moments

This modification decreased the torque requirements of the knee servos, enabling Pavo to stand without issues and begin performing the movements detailed in section 4.2 below.

4.2. Results

With Pavo successfully built and control theory coded and implemented, Pavo's ability to combine the appropriate sets and execute a gait cycle was tested, this can be seen in video Figure 39 for walking forward (xMod = 100, yMod =0) and stepping right (xMod = 0, yMod = 100).

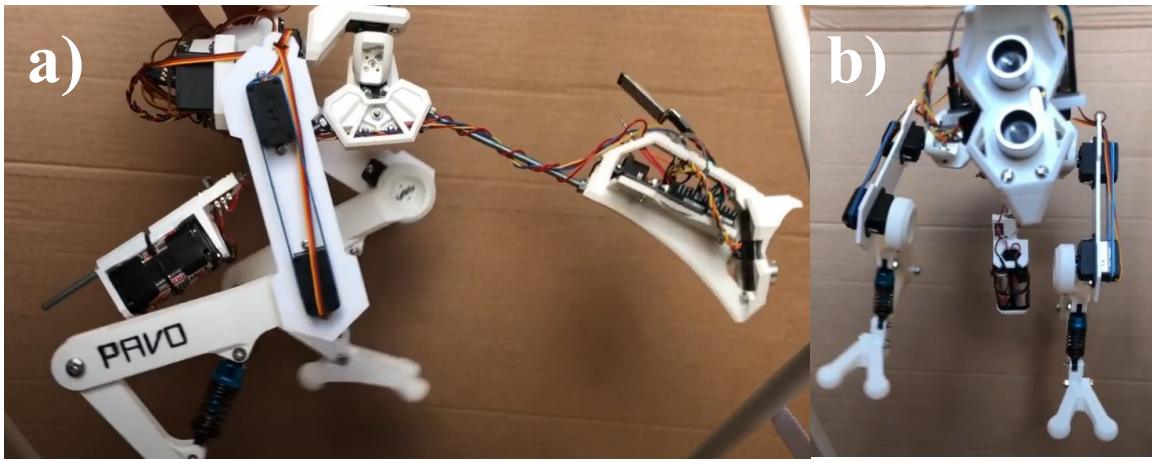


Figure 40: [VIDEO] A demonstration of the cyclic footstep functionality with slowed down and exaggerated movements for a) walking front and b) stepping right (full links can be found in the appendix (A.1) if embedded links are not functioning)

Pavo's ability to respond to all three external stimuli was also tested successfully, with video Figure 40 demonstrating Pavo's ability to move in the direction of fall by utilising the onboard fuzzy logic lookup table, video Figure 41 demonstrating its ability to sense an obstacle and move backwards, and video Figure 42 demonstrating a successful response to wireless external input from the controller. In all the aforementioned Figures, the speed of the cycle has been reduced and movements exaggerated to allow the observation of the distinct cycle stages.

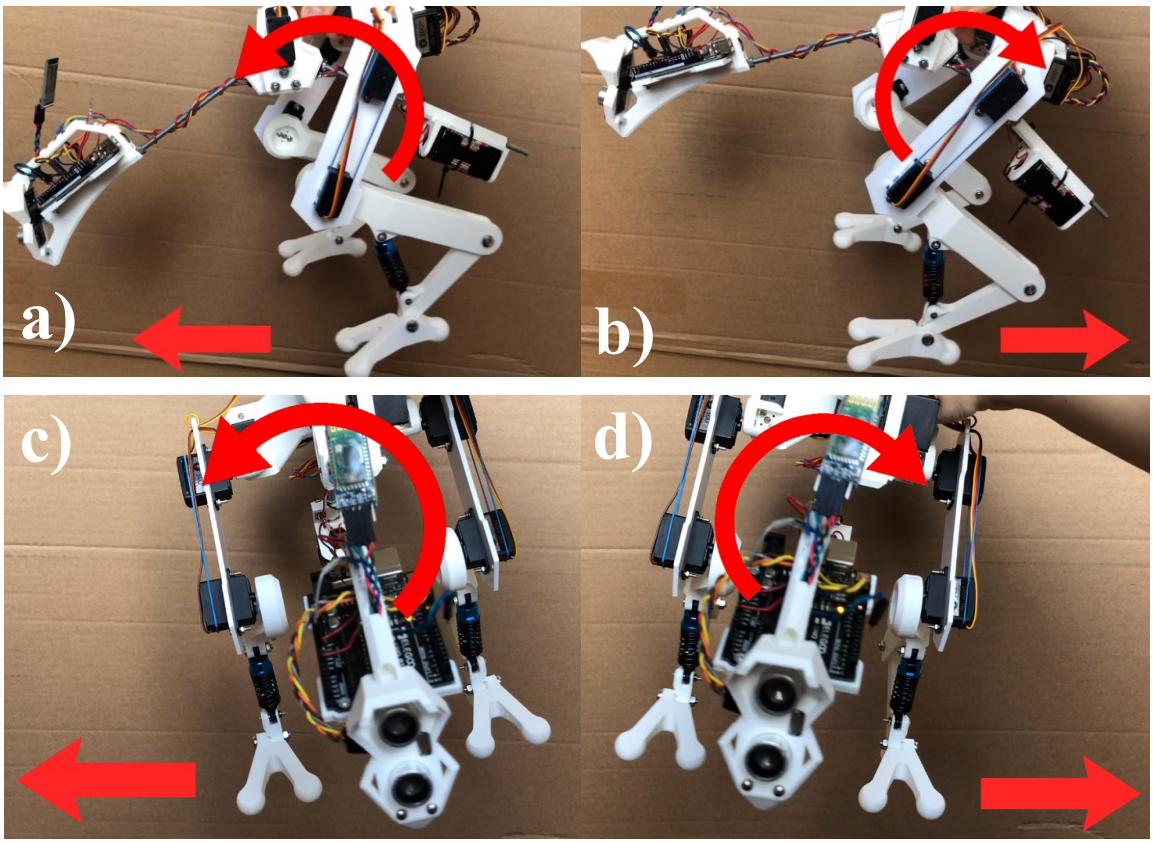


Figure 41: [VIDEO] A demonstration of Pavo's ability to respond accordingly to angular position and come to rest when rebalanced a) Tilting forwards and walking forward, b) Tilting backwards and walking backwards, c) Tilting right and walking right, d) Tilting left and walking left

Figure 42: [VIDEO] A demonstration of Pavo's ability to respond to a sensed obstacle and begin a backwards movement and stop when at a safe distance

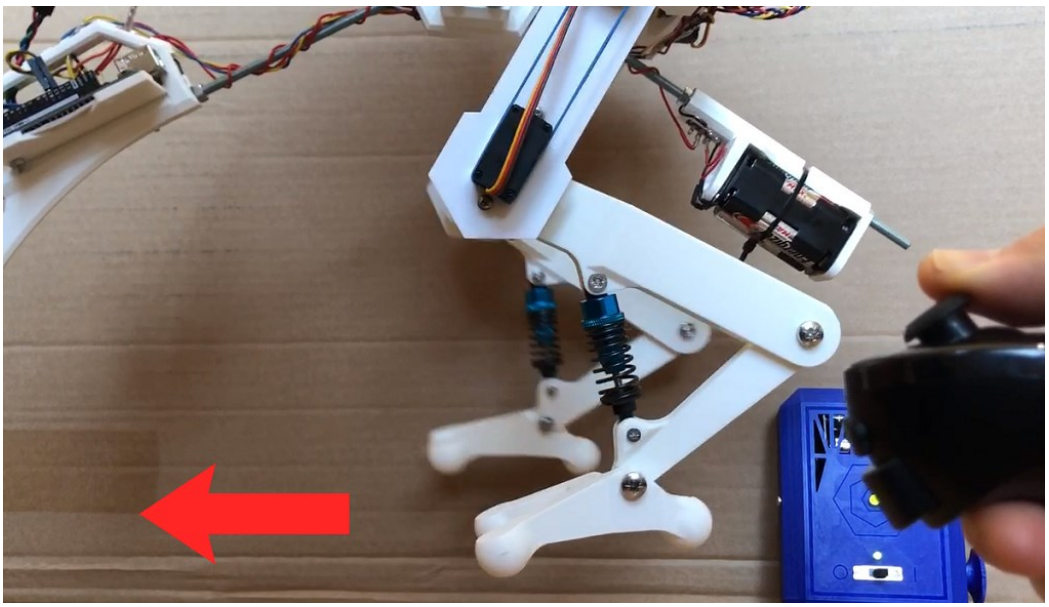


Figure 43: [VIDEO] A demonstration of Pavo's ability to respond to controller input appropriately

As mentioned above, the servos were unable to provide the adequate torque required to mimic the walking of a quail as seen in Figure 6. As such, the angular ranges of movement were tuned down to a point where the servos were able to execute the positions written to them without skipping/failing.

These smaller steps allowed Pavo to walk in a "shuffle", moving across a surface in a constantly stable way, maintaining double support throughout the majority of the gait cycle instead of the planned 25%. This, unfortunately, prevented the fuzzy logic system from seeing meaningful implementation in balancing Pavo and preventing falls, further discussed in section 4.3.5.

The average speed was then obtained by measuring the time taken for Pavo to traverse a distance of 70cm, this can be seen in video Figure 43.

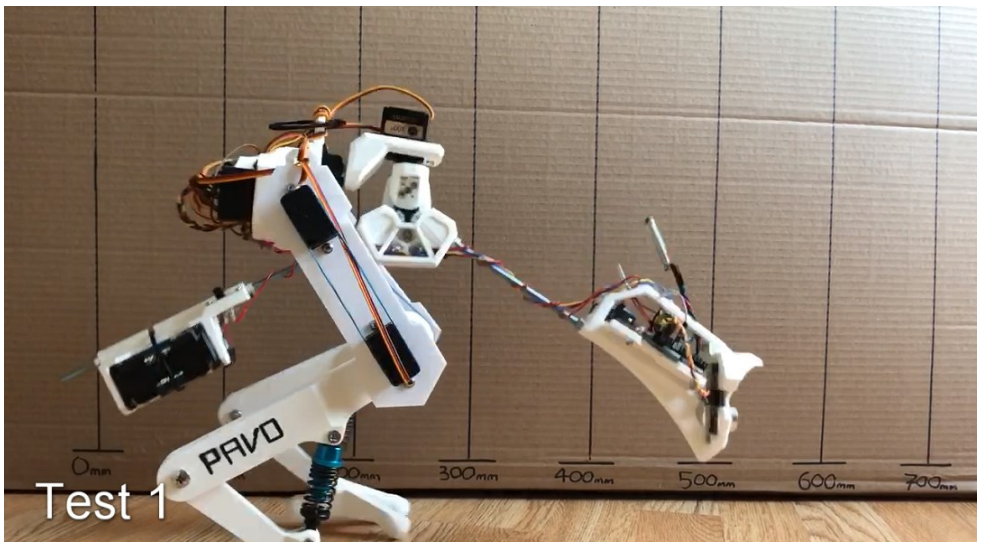


Figure 44: [VIDEO] A demonstration of Pavo walking 70cm

This timed test was repeated a total of six times and durations averaged. The average speed obtained was 29.371mm/s (individual test results can be found in the appendix, A.7).

The consistency of the gait cycle and its speed were also tested by overlaying the six tests to observe the variations between them. This can be seen in video Figure 44.

Figure 45: [VIDEO] Six speed tests overlayed

It can be seen that Pavo is fairly consistent in maintaining its speed, and saw a maximum variance of 28.6% between its shortest and longest run. This test was repeated from a top-down view to assess Pavo's ability to maintain a straight course, as seen in video Figure 45.

Figure 46: [VIDEO] Five directional tests overlayed

From Figure 45, It can be seen that Pavo is also fairly consistent in its direction, not veering off by more than a few degrees over the course of the test, however, a tendency to turn left was observed, this may be a result of an underpowered servo or a slight incline of the surface. Directional consistency may be improved with the implementation of a turning function and usage of data from the currently unutilised onboard magnetometer, further discussed in section 4.3.6.

4.3. Issues, Improvements and Future Work

Throughout the development and testing of the robot, many issues were identified. These issues and their possible solutions/suggestions for future work are as follows.

4.3.1. Servo Torques and Power

In its current state, one of the main imitations of Pavo is its servos. They are unable to provide the torques required to enable the full implementation and exploration of the control systems developed.

A simple way to reduce the torques required is to reduce the size of the robot, reducing moments of inertia and weight simultaneously. However, this reduces its effectiveness at emulating a real-world implementation of the robot which would likely be larger instead of smaller and does not address the issue directly.

Utilising better actuators such as those used in the robots mentioned in section 1.1 would improve Pavo's ability to perform larger movements. This may, however, increase the cost of the robot, this cost can be alleviated by only upgrading certain servos that require the most torque/are currently the most likely to be overloaded (i.e., the knee and hip joints).

Further research and implementation of compliance-based mechanisms to store energy during the walking gait such as in section 4.1.2 and in work done by Badri-Spröwitz et al. (2022) would further reduce actuator loads, it may also attenuate undesirable oscillations and even decrease power draw.

Finally, reducing the weights of various parts of the robot with better design would also reduce the torques required to actuate components, methods of reducing weight are elaborated on in section 4.3.2 below.

4.3.2. Improved Construction of the Robot

The inability to actuate larger movements, resulting in the shuffling motion of Pavo may also be attributed to the compliance/deformations while walking due to its construction,

with the entire construction bending to accommodate a new position with both legs still on the ground instead of swinging one forward as intended.

This may be improved by utilising more rigid materials and improving the design. Different manufacturing methods such as metal CNC machining and direct metal laser sintering (DMLS) may be utilised to realise this. Design methods such as generative design (GD) or topology optimisation may also be utilised to reduce weight and increase rigidity such as in Junk et al. (2018) or Rajput et al. (2021), where GD was utilised to redesign a prosthetic leg as seen in Figure 46.

Figure 47: A generatively designed calf prosthetic, reducing material cost and weight while maintaining required strength (Rajput et al. 2021)

The joints connecting components may also be improved to reduce this unintentional flexibility, for example, by replacing the current joints with better, more rigid ball bearing-based joints.

Power solutions such as lithium-ion batteries with better power-to-weight ratios may also prove to be effective methods of decreasing overall weight.

The spring damper integrated into Pavo's legs could also be tuned to better attenuate oscillations that interfere with IMU data. As it stands, the spring dampers are too stiff and may be considered fully rigid, providing negligible damping effects.

4.3.3. Computing Power

A limitation encountered throughout the development of Pavo was the limited computational capacity of the Arduino UNO, with its computing speed, flash memory capacity and static random-access memory (SRAM) capacity all bottlenecking the implementation of the control system developed.

One way of circumventing this issue is to improve/optimise the code to make it run more efficiently and to utilise less memory and SRAM. With code of this complexity, there are likely many avenues to explore to accomplish this such as reusing arrays and utilising more efficient algorithms to calculate/generate data. As mentioned in section 3.1, the code is currently structured in a compartmentalised fashion, with functions working independently to use, process and execute data received from separate functions higher in the chain. This is to facilitate the ease of improving and updating the code, however, once developed to a satisfactory degree, the code may be rewritten to utilise the data in a much more efficient and cohesive manner, removing the need for the "chain" of functions that may slow the program down and limit the robots ability to function.

This does not however remove the hard limitations such as the Arduino's 32 Kilobyte flash memory that limits the size (and therefore accuracy) of the fuzzy logic look-up table and the 2 Kilobytes of SRAM that limit the number of transitionary sets that can be stored (reducing movement smoothness). New hardware must be considered for this, boards such as the more powerful Arduino DUE, with 512 Kilobyte flash memory and 96 Kilobyte SRAM, or a Raspberry Pi as considered in section 2.2 could improve computing power without sacrificing size, a size comparison of these solutions can be seen in Figure 47.

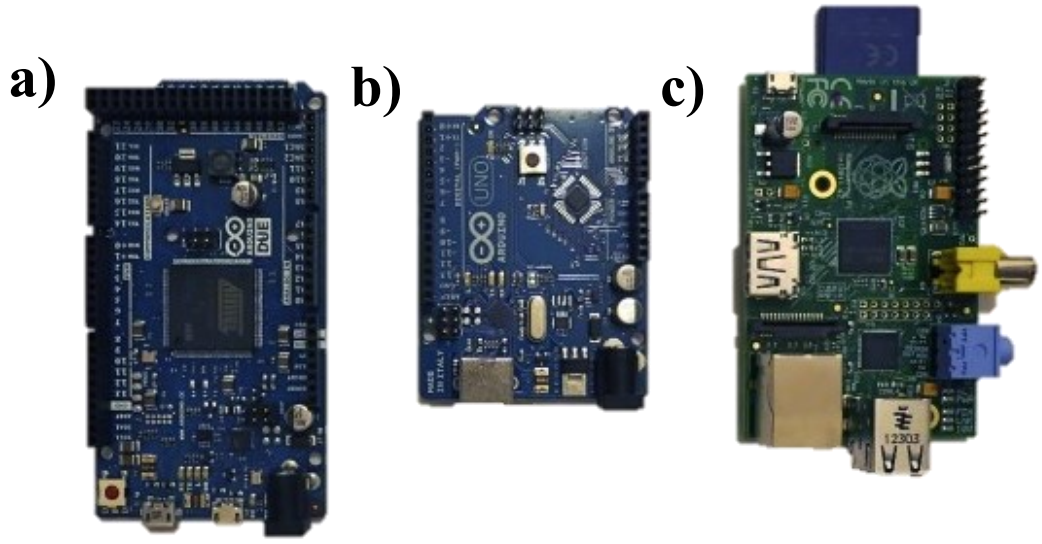


Figure 48: A size comparison of: a) Arduino DUE, b) The currently implemented Arduino UNO, c) Raspberry Pi (Senese 2012)

The computational capacity may also be improved by further offloading calculations to an external board/computer, and sending/receiving the data wirelessly, further discussed in section 4.3.4 below.

4.3.4. Bluetooth Communication

The current implementation of one-way Bluetooth communication stands to be greatly improved. A proper Bluetooth data packet strategy may be implemented to facilitate bi-directional data transfer at much higher speeds and volumes.

This allows the sharing of processing power between the two Arduinos as opposed to the current configuration where all calculations are performed on the onboard Arduino alone. This may effectively double the memory and SRAM available if implemented properly, without needing to upgrade hardware. For example, the robot could send obstacle data and raw IMU data to the controller-side Arduino, and have it use that data to calculate the next movement set required before sending that back to the robot to execute.

This concept may be explored further by offloading computation onto more powerful systems such as personal computers that do not have to be carried or powered by the robot.

4.3.5. Fuzzy Logic

As mentioned above, due to the limitations of the actuators utilised, the fuzzy logic system planned could not be fully realised. While it has been successfully implemented and shown to respond appropriately to external data, the reduced footstep ranges did not allow Pavo to truly mimic the cyclic “falling and catching” of a natural quail gait cycle.

Even with adequate servos, it is still expected that more testing and refining of the fuzzy logic system is required before the robot is truly able to reliably mimic the gait cycle of a quail. With weights and membership functions needing to be adjusted and different approaches possibly explored, such as the fuzzy logic returning a directional acceleration instead of a directional speed.

Simulations of simplified models of the robot may also be performed to provide a better starting point that may then be further refined in the physical model.

The implemented fuzzy logic currently accepts two inputs and produces one output, resulting in a 2-dimensional array look-up table, more inputs and outputs could also be explored with improved hardware allowing the storing of larger, higher-dimensional arrays, one example of an extra fuzzy logic input is described in section 4.3.6.

4.3.6. Environmental Data/Feedback

For a robot to traverse an environment effectively, comprehensive environmental data is needed, a logical improvement would therefore be increasing/improving the data that the robot can gather from its environment, and how it reacts to it.

Firstly, obstacles are currently avoided by simply adding a bias to move backwards upon detection of an object, this may be improved by programming a system to allow the robot to actively change its trajectory to avoid the object, it may also be possible to have the robot stop and survey its surroundings before choosing the best path forward to avoid the obstacle.

Sensory components could also be added/upgraded, such as additional ultrasonic sensors, camera systems, servo position/torque feedback, or extra IMUs, as described in section 1.1 above. Location/heading data such as from the currently unused magnetometer or a GPS may also improve directional consistency and allow the robot to navigate to set locations/in set directions.

This environmental data may then be utilised as additional inputs for the fuzzy logic system and produce an improved directional output, such as in the work by Mishra et al. (2022), where ultrasonic sensor data was utilised as fuzzy logic inputs for a robot to avoid obstacles.

4.3.7. Gait/Walking Cycle

Pavo is currently susceptible to inclined surfaces and requires a surface with an appropriate friction to walk. These issues may be alleviated by improving aspects of the gait cycles, along with the use of improved environmental data as described above.

Firstly, more work could be done in programming the walk cycles based on the gait of the quail, with a more faithful representation of its movements achievable. Conversely, as Pavo is not an exact replica of a quail, work may also be done to optimise the cycle for the built configuration instead, with improvements such as rhythmic rolling of the body to counteract the sideways tilts caused by the leg swings.

The movements are currently broken down into three sets (“empty”, “front/back” and “left/right”) that are combined with different weights to produce a footstep, future work may use this easily expandable framework to break down complex movements into further sub-categories to further refine and improve the control of the CPG. For example, breaking down the “empty” set into separate footstep and counterbalance movements, and having the counterbalance movements be informed by IMU data to counteract more dynamic, unexpected perturbances. Another example could be a new sub-set concerning rhythmic asymmetrical body yaw, this may be adjusted to allow the robot to veer in a desired direction while walking forwards or backwards.

5. Conclusion

A literature review of bipedal robots was conducted, concluding that bird-based bipedal robots present a promising new approach to the bipedal problem, resulting in lower CoMs and better stability. It was also found that many control theories utilised in the control of current bipedal robots are based upon oftentimes unrealistic simplifications/assumptions and are computationally expensive.

Consequently, a low-cost bipedal robot based on a quail was designed and built utilising the minimum DoFs needed for bipedal movement and balance. A novel control theory utilising fuzzy logic and based upon an experimentally obtained quail gait cycle was developed, coded and implemented, with emphasis placed on upgradability to facilitate future work.

Issues throughout its development were identified and remedied where possible, chief among these include:

- The servos being underpowered for the movements initially planned, demonstrating the importance of compliant ligament-like components in bipeds. This resulted in Pavo only being capable of taking small steps, preventing the fine-tuning of the fuzzy logic system to operate as intended.
- The overly-compliant construction of Pavo preventing positions from being enacted as initially planned.
- The Arduino UNO's speed, SRAM and memory limitations reducing the speed and smoothness of the footstep cycles, and also limiting the accuracy of the fuzzy logic implementation.

In conclusion, Pavo was able to walk without falling in a "shuffling" fashion at an average speed of 29.371mm/s. Pavo was shown to successfully utilise non-rhythmic external data to produce adaptive cyclic footstep patterns, autonomously responding to IMU data, obstacle sensing, and operator inputs, indicative of a successful implementation of the control theory developed.

References

Abourachid, Anick, Remi Hackert, Marc Herbin, Paul A. Libourel, François Lambert, Henri Gioanni, Pauline Provini, Pierre Blazevic, and Vincent Hugel. 2011. “Bird Terrestrial Locomotion as Revealed by 3D Kinematics.” *Zoology* 114(6):360–68. doi: 10.1016/J.ZOOL.2011.07.002.

Aithal, Chandana N., P. Ishwarya, S. Sneha, C. N. Yashvardhan, and K. v. Suresh. 2021. “Design of a Bipedal Robot.” *Lecture Notes in Electrical Engineering* 752 LNEE:55–67. doi: 10.1007/978-981-16-0443-0_5.

Andrada, Emanuel, Christian Rode, Yefta Sutedja, John A. Nyakatura, and Reinhard Blickhan. 2014. “Trunk Orientation Causes Asymmetries in Leg Function in Small Bird Terrestrial Locomotion.” *Proceedings of the Royal Society B: Biological Sciences* 281(1797). doi: 10.1098/rspb.2014.1405.

Andrews, Christopher. 2015. “PGMWrap Library for AVR and Arduino Boards.” Retrieved April 17, 2022 (https://github.com/Chris--A/PGMWrap).

Badri-Spröwitz, Alexander, Alborz Aghamaleki Sarvestani, Metin Sitti, and Monica A. Daley. 2022. *BirdBot Achieves Energy-Efficient Gait with Minimal Control Using Avian-Inspired Leg Clutching*. Vol. 7. doi: 10.1126/scirobotics.abg4055.

Bae, Hyoin, and Jun Ho Oh. 2018. “Biped Robot State Estimation Using Compliant Inverted Pendulum Model.” *Robotics and Autonomous Systems* 108:38–50. doi: 10.1016/J.ROBOT.2018.06.004.

Bartoš, Michal, Vladimír Bulej, Martin Bohušík, Ján Stancek, Vitalii Ivanov, and Peter Macek. 2021. “An Overview of Robot Applications in Automotive Industry.” *Transportation Research Procedia* 55:837–44. doi: 10.1016/J.TRPRO.2021.07.052.

Bruemmer, David J., and Mark S. Swinson. 2003. “Humanoid Robots.” *Encyclopedia of Physical Science and Technology* 401–25. doi: 10.1016/B0-12-227410-5/00317-3.

Brusatte, Stephen L., Jingmai K. O’Connor, and Erich D. Jarvis. 2015. “The Origin and Diversification of Birds.” *Current Biology* 25(19):R888–98. doi: 10.1016/J.CUB.2015.08.003.

Carlos De Pina Filho, Armando, Aloísio Carlos De Pina, and Yuri dos Santos Mota. 2010. “RESEARCH ON BIPEDAL ROBOTS APLLIED TO SOCIETY.”

Carolo, Lucas. 2020. “Arduino vs Raspberry Pi: The Differences.” Retrieved April 17, 2022 (https://all3dp.com/2/arduino-vs-raspberry-pi/).

Chang, Lin, Songhao Piao, Xiaokun Leng, Zhicheng He, and Zheng Zhu. 2020. “Inverted Pendulum Model for Turn-Planning for Biped Robot.” *Physical Communication* 42:101168. doi: 10.1016/J.PHYCOM.2020.101168.

Douglas, Brian. 2021. “Fuzzy Logic Part 3: Fuzzy Logic Examples.” Retrieved April 16, 2022 (https://uk.mathworks.com/videos/fuzzy-logic-part-3-design-and-applications-of-a-fuzzy-logic-controller-1631704221859.html).

Earl, Bill. 2012. “Adafruit PCA9685 16-Channel Servo Driver.” Retrieved April 17, 2022 (https://learn.adafruit.com/16-channel-pwm-servo-driver/).

Ficht, Grzegorz, and Sven Behnke. 2021. “Bipedal Humanoid Hardware Design: A Technology Review.” doi: 10.48550/arXiv.2103.04675.

Ficht, Grzegorz, Hafez Farazi, André Brandenburger, Diego Rodriguez, Dmytro Pavlichenko, Philipp Allgeuer, Mojtaba Hosseini, and Sven Behnke. 2018. “NimbRo-OP2X: Adult-Sized Open-Source 3D Printed Humanoid Robot; NimbRo-OP2X: Adult-Sized Open-Source 3D Printed Humanoid Robot.” *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*. doi: 10.0/Linux-x86_64.

Finlayson, Clive. 2005. “Biogeography and Evolution of the Genus Homo.” *Trends in Ecology & Evolution* 20(8):457–63. doi: 10.1016/J.TREE.2005.05.019.

Grylls, Bethan. 2018. “Ocado’s Robot Swarm.” Retrieved April 24, 2022 (https://www.newelectronics.co.uk/content/news/ocado-s-robot-swarm).

Hu, Jianjuen J., and Arthur C. Smith. 2000. “Stable Locomotion Control of Bipedal Walking Robots: Synchronization with Neural Oscillators and Switching Control.”

Hunt, Kevin D. 2015. “Bipedalism.” *Basics in Human Evolution* 103–12. doi: 10.1016/B978-0-12-802652-6.00008-6.

Ijspeert, Auke Jan. 2008. “Central Pattern Generators for Locomotion Control in Animals and Robots: A Review.” *Neural Networks* 21(4):642–53. doi: 10.1016/J.NEUNET.2008.03.014.

Junk, Stefan, Benjamin Klerch, Lutz Nasdala, and Ulrich Hochberg. 2018. “Topology Optimization for Additive Manufacturing Using a Component of a Humanoid Robot.” *Procedia CIRP* 70:102–7. doi: 10.1016/J.PROCIR.2018.03.270.

Kaur, Arshdeep, and Amrit Kaur. 2012. “Comparison of Mamdani-Type and Sugeno-Type Fuzzy Inference Systems for Air Conditioning System.” *International Journal of Soft Computing and Engineering (IJSCE)* 2(2). doi: 10.35940/ijsce.

Ksepka, Daniel T. 2022. “Evolution of Birds.” *Sturkie’s Avian Physiology* 83–107. doi: 10.1016/B978-0-12-819770-7.00009-8.

Lepora, Nathan F., Anna Mura, Michael Mangan, Paul F. M. J. Verschure, Marc Desmulliez, and Tony J. Prescott. 2016. “The Natural Bipeds, Birds and Humans: An Inspiration for Bipedal Robots.” *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 9793:V–VIII. doi: 10.1007/978-3-319-42417-0.

Lin, Bintian, Qingwen Zhang, Feng Fan, and Shizhao Shen. 2021. “Reproducing Vertical Human Walking Loads on Rigid Level Surfaces with a Damped Bipedal Inverted Pendulum.” *Structures* 33:1789–1801. doi: 10.1016/J.ISTRUC.2021.05.048.

van der Linden, Marietta. 2011. “Gait Analysis, Normal and Pathological Function, 2nd Ed. J. Perry, J.M. Burnfield, Slack Inc., 576 Pages, ISBN 978-1-55642r-R766-4.” *Physiotherapy* 97(2):180. doi: 10.1016/J.PHYSIO.2010.05.007.

Liu, Chuan, and Ruiming Qian. 2019. “An Action Generator for Small Humanoid Robot Based on Inverse Kinematics.” in *The 4th International Conference on Control and Robotics Engineering : ICCRE 2019 : April 20-23, 2019, Nanjing, China*.

Liu, George H. Z., Michael Z. Q. Chen, and Yonghua Chen. 2019. “When Joggers Meet Robots: The Past, Present, and Future of Research on Humanoid Robots.” *Bio-Design and Manufacturing* 2(2):108–18. doi: 10.1007/s42242-019-00038-7.

Lohman, Everett B., Kanikkai Steni Balan Sackiriyas, and R. Wesley Swen. 2011. “A Comparison of the Spatiotemporal Parameters, Kinematics, and Biomechanics between Shod, Unshod, and Minimally Supported Running as Compared to Walking.” *Physical Therapy in Sport* 12(4):151–63. doi: 10.1016/J.PTSP.2011.09.004.

Madison, Dave. 2021. “NintendoExtensionCtrl Arduino Library.” Retrieved April 15, 2022 (https://github.com/dmadison/NintendoExtensionCtrl).

Maiorino, Andrea, and Giovanni Gerardo Muscolo. 2020. “Biped Robots With Compliant Joints for Walking and Running Performance Growing.” *Frontiers in Mechanical Engineering* 6. doi: 10.3389/fmech.2020.00011.

McGhee, R. B. 1968. “Some Finite State Aspects of Legged Locomotion.” *Mathematical Biosciences* 2(1–2):67–84. doi: 10.1016/0025-5564(68)90007-2.

Mishra, Krishna Anurag, Shivam Agarwalla, Daiwik Mishra, Anshuman Samal, Ashwani Kumar, P. Chandrasekhar, Abhishek Sharma, and Anish Pandey. 2022. “Fuzzy

Logic Controlled Autonomous Quadruped Robot." *Materials Today: Proceedings*. doi: 10.1016/J.MATPR.2022.02.239.

Nguyen, Xuan Tien, Dang Hung Nguyen, Huy Hung Nguyen, Nhut Phuong Tong, Thanh Phuong Nguyen, and Tan Tien Nguyen. 2020. "Balancing Walking Gait for Small Size Humanoid Robot by Using Movable Mass." *International Conference on Advanced Mechatronic Systems, ICAMechS* 2020-December:13–16. doi: 10.1109/ICAMechS49982.2020.9310109.

Nyakatura, J. A., E. Andrada, N. Grimm, H. Weise, and M. S. Fischer. 2012. "Kinematics and Center of Mass Mechanics During Terrestrial Locomotion in Northern Lapwings (Vanellus Vanillas, Charadriiformes)." *Journal of Experimental Zoology Part A: Ecological Genetics and Physiology* 317(9):580–94. doi: 10.1002/jez.1750.

Park, Hae Won, Koushil Sreenath, Jonathan W. Hurst, and Jessy W. Grizzle. 2011. "Identification of a Bipedal Robot with a Compliant Drivetrain: Parameter Estimation for Control Design." *IEEE Control Systems* 31(2):63–88. doi: 10.1109/MCS.2010.939963.

Rajput, Srijan, Himanshu Burde, Udit Suraj Singh, Hridik Kajaria, and Ranjeet Kumar Bhagchandani. 2021. "Optimization of Prosthetic Leg Using Generative Design and Compliant Mechanism." *Materials Today: Proceedings* 46:8708–15. doi: 10.1016/J.MATPR.2021.04.026.

Reher, Jacob, Wen-Loong Ma, and Aaron D. Ames. 2019. "Dynamic Walking with Compliance on a Cassie Bipedal Robot." doi: 10.48550/arXiv.1904.11104.

Reis, João, Nuno Melão, Juliana Salvadorinho, Bárbara Soares, and Ana Rosete. 2020. "Service Robots in the Hospitality Industry: The Case of Henn-Na Hotel, Japan." *Technology in Society* 63:101423. doi: 10.1016/J.TECHSOC.2020.101423.

Senese, Mike. 2012. "ARDUINO UNO AND DUE SIZE COMPARISON (WITH RASPBERRY PI AND FIO, TOO)." Retrieved April 19, 2022 (http://www.mikesenese.com/DOIT/2012/10/arduino-uno-and-due-size-comparison-with-raspberry-pi-and-fio-too/).

Siepert, Bryan. 2020. "Adafruit TDK InvenSense ICM-20948 9-DoF IMU." Retrieved April 17, 2022 (https://learn.adafruit.com/adafruit-tdk-invensense-icm-20948-9-dof-imu).

Singh, Harpreet, Madan M. Gupta, Thomas Meitzler, Zeng-Guang Hou, Kum Kuma Garg, Ashu M. G. Solo, and Lotfi A. Zadeh. 2013. "Editorial Real-Life Applications of Fuzzy Logic." *Advances in Fuzzy Systems* 2013. doi: 10.1155/2013/581879.

Sobhan, P. V. S., G. v. Nagesh Kumar, M. Ramya Priya, and B. Venkateswara Rao. 2009. “Look up Table Based Fuzzy Logic Controller for Unmanned Autonomous Underwater Vehicle.” *ACT 2009 - International Conference on Advances in Computing, Control and Telecommunication Technologies* 497–501. doi: 10.1109/ACT.2009.128.

Takanishi, Atsuo. 2019. “Historical Perspective of Humanoid Robot Research in Asia.” *Humanoid Robotics: A Reference* 35–52. doi: 10.1007/978-94-007-6046-2_145.

Tsagarakis, N. G., D. G. Caldwell, F. Negrello, W. Choi, L. Baccelliere, V. G. Loc, J. Noorden, L. Muratore, A. Margan, A. Cardellino, L. Natale, E. Mingo Hoffman, H. Dallali, N. Kashiri, J. Malzahn, J. Lee, P. Kryczka, D. Kanoulas, M. Garabini, M. Catalano, M. Ferrati, V. Varricchio, L. Pallottino, C. Pavan, A. Bicchi, A. Settimi, A. Rocchi, and A. Ajoudani. 2017. “WALK-MAN: A High-Performance Humanoid Platform for Realistic Environments.” *Journal of Field Robotics* 34(7):1225–59. doi: 10.1002/rob.21702.

Vaughan, Christopher L. 2003. “Theories of Bipedal Walking: An Odyssey.” *Journal of Biomechanics* 36(4):513–23. doi: 10.1016/S0021-9290(02)00419-0.

Warnakulasooriya, Sujan, Amin Bagheri, Nathan Sherburn, and Madhavan Shanmugavel. 2012. “Bipedal Walking Robot - A Developmental Design.” *Procedia Engineering* 41:1016–21. doi: 10.1016/j.proeng.2012.07.277.

Xie, Ye, Bin Lou, Anhuan Xie, and Dan Zhang. 2020. “A Review: Robust Locomotion for Biped Humanoid Robots.” *Journal of Physics: Conference Series* 1487(1). doi: 10.1088/1742-6596/1487/1/012048.

Zadeh, L. A. 1965. “Fuzzy Sets.” *Information and Control* 8(3):338–53. doi: 10.1016/S0019-9958(65)90241-X.

Appendix

A.1 Video Figure Links

Figure 39: https://www.youtube.com/embed/ajH8I8Yix_Y?start=0&end=22

Figure 40: https://www.youtube.com/embed/ajH8I8Yix_Y?start=24&end=59

Figure 41: https://www.youtube.com/embed/ajH8I8Yix_Y?start=61&end=84

Figure 42: https://www.youtube.com/embed/ajH8I8Yix_Y?start=86&end=124

Figure 43: https://www.youtube.com/embed/ajH8I8Yix_Y?start=216&end=255

Figure 44: https://www.youtube.com/embed/ajH8I8Yix_Y?start=126&end=159

Figure 45: https://www.youtube.com/embed/ajH8I8Yix_Y?start=161&end=210

A.2 Parts list and Costing

Item	Description/Specifications	Links	Amount	Price
Arduino Uno equivalent (Elegoo Branded)	32Kb of flash memory, 2Kb of SRAM, a clock speed of 16MHz, 14 digital IO pins, 6 analogue IO pins, 5V operation.	https://www.amazon.co.uk/gp/product/B01EWOE0UU/ref=ppx_yo_dt_b_asin_title_o06_s00?ie=UTF8&psc=1	2	£ 16.64
Adafruit 16-Channel 12-bit PWM Servo Driver	Utilises I2C communication, taking up only two pins on the Arduino, drives and powers up to 16 servos simultaneously.	https://thepihut.com/products/adafruit-16-channel-12-bit-pwm-servo-driver-i2c-interface-pca9685?variant=27740507729&currency=GBP&utm_medium=product_sync&utm_source=google&utm_content=sag_organic&utm_campaign=sag_organic	1	£ 13.00
ICM-20948 9-DoF Inertial measurement unit	3 axes of accelerometer data, gyroscopic data, and magnetometer data. The IMU is accurate to ±250 degrees per second for the gyroscope, ±2g for the accelerometer and ±4900µT for the magnetometer.	https://thepihut.com/products/adafruit-tdk-invensense-icm-20948-9-dof-imu-mpu-9250-upgrade?ref=isp_rel_prd&isp_ref_pos=3	1	£ 13.00
HC-05 Bluetooth module	Configured with one master and one slave.	https://thepihut.com/products/hc-05-bluetooth-module	2	£ 8.00
HC-SR04 Ultrasonic Rangefinder	Capable of sensing objects within a range of 2cm to 400cm, at an accuracy of approximately 3mm.	https://thepihut.com/products/ultrasonic-distance-sensor-hcsr04	1	£ 2.00
Nintendo “Nunchuck”	Capable of generating gyroscopic pitch and roll outputs and outputs from its 2-axis joystick and two buttons, Interfaces via I2C	https://www.amazon.co.uk/gp/product/B01CJ9J7T4/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&th=1	1	£ 6.99

Servos SER0056	Capable of producing a maximum continuous torque of 0.55 kgf·cm, up to 300° of motion	https://www.mouser.co.uk/ProductDetail/DFRobot/SER0056?qs=sGAEpiMZZMv0NwlthflBiyy1cZtCJtiGe7ORwfYwunA%3D	9	£ 41.49
AA Battery	Pack of 12, 1.5V, 3Ah Capacity, with four batteries, supplies 6 volts to the electronics	https://uk.rs-online.com/web/p/aa-batteries/1974299	12	£ 7.86
AA Battery Holders	Holds four AA Batteries	https://uk.rs-online.com/web/p/battery-holders/6119605	2	£ 1.40
RC Damper Suspension	Hobby radio-controlled car suspension system.	https://www.amazon.co.uk/gp/product/B00RF2W5OA/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&psc=1	1	£ 10.01
STEMMA QT Proprietary cable	To allow stronger, neater connections to the IMU	https://thepihut.com/products/stemma-qt-qwiic-jst-sh-4-pin-cable-100mm-long	1	£ 0.80
Assorted wires, resistors, etc.	Provided by Electronics workshop	N/A	N/A	N/A
3D printed/laser cut parts	Provided by Design Workshop	N/A	N/A	N/A
			Total	£ 121.19

A.3 Software Utilised

Software:	Utilised for:
Fusion 360	Design of Pavo
Solidworks	Engineering drawings of Pavo
MATLAB	Coding of fuzzy logic and code to generate Arduino compatible lookup table
Arduino IDE	Code editor and compiler for Arduino UNO
Lightworks	Video editor utilised for video Figures
Draw.io	To produce diagrams/charts
Fritzing	To produce circuit diagrams
Cura	To slice 3D files for 3D printing
PrusaSlicer	To analyse and verify parts before sending to 3D print

A.4 Risk Assessment

Due to the highly practical and hands-on nature of this project, where many manufacturing facilities and workshops were required to be utilised to realise the final tangible product, training was undergone prior to using their respective facilities to ensure a high standard of health and safety was upheld at all times.

An induction was held for the design workshop, wherein appropriate precautions were underlined before using the Pillar drill and bandsaw, and various other tools and equipment on site.

An induction was held for the electronics workshop, where a guided tutorial on soldering was performed, ensuring good practices and safety precautions were followed. An introduction to the use of an oscilloscope and signal generators was also performed, although not utilised in this project.

An induction was also performed to learn how to operate the self-service 3D printers in the design workshop, although this was also not utilised for the project.

Prior to utilising any of the aforementioned facilities for the project, appropriate method statements and risk assessments were undertaken and approved before use to ensure the safety of the individual utilising the equipment and also personnel around and in the vicinity. This results in proper assessment of possible hazards in these environments, such as hot soldering irons causing fires, laser cutters causing blindness or hazardous fumes, reckless band saw usage causing injury, etc. These forms can be found below.

A.4.1 Risk Assessment Forms

A.4.1.1 Design workshop Method Statement

Faculty of Engineering and the Environment	**Method Statement**
Title **Design workshop activities for the building of a Bipedal Robot.**	
Location of Activity *Tizard Building 13, Room 1055*	**Date** **8/3/2022**
Assessor *Ryan Khoo Yeap Hong* *ryhk1a18 30480183*	**Contact Details** *ryhk1a18@soton.ac.uk* *+44 0749 376 4830*
Supervisor Dr Suleiman Sharkh	**Contact Details** *S.M.Sharkh@soton.ac.uk*

Introduction / Overview.

Background description to the project. What will you achieve? How will you do this? Why is this required?

A Bipedal robot is to be built, to do so, various workshop work will be done to enable the completion of the robot. This will be done in the Design workshop by utilising its tools and resources to: Laser cut acrylic/ply-wood pieces, assemble and modify them. Plastic 3D-printed parts may also be assembled and modified. This may involve simple drilling, sanding, cutting and deburring activities.

Description of Task and how it will be carried out.

Including any diagrams, materials, samples and equipment to be used as applicable.

The main activity that will be carried out will be laser cutting parts and assembling them. This will involve placing material into the laser cutters in the workshop and operating them safely. Besides this, parts will also be assembled, with slight modifications possibly required, using the various tool available in the workshop, such as drilling, cutting and sawing.

Control Measures including training, PPE

Identify significant hazards and actions/control measures to be taken.

Hazards present include machinery used, Pillar drills, band saws and laser cutters, among other assorted tools. These pose a threat to safety if handled carelessly or misused, for example, potentially serious cuts from band saws, or blindness from stray lasers. Covered shoes are worn at all times to protect from falling items, and safety glasses will be utilised while laser cutters are in operation.

An official induction for the workshop was also attended to ensure adequate knowledge of how to utilise the various tools and equipment available prior to unsupervised activity.

Emergency Arrangements

-

Additional persons involved in activity

-

A.4.1.2 Design workshop Risk Assessment

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

Risk Assessment			
Risk Assessment for the activity of	**Design Workshop activities**	**Date**	**8/3/2022**
Unit/Faculty/Directorate	**Mechanical Engineering**	**Assessor**	**Ryan Khoo**
Line Manager/Supervisor	***Dr Suleiman Sharkh***	**Signed off**	

PART A											
(1) Risk identification			**(2) Risk assessment**				**(3) Risk management**				
Hazard	**Potential Consequences**	**Who might be harmed** (user; those nearby; those in the vicinity; members of the public)	**Inherent** Likelihood	Impact	Score	**Control measures (use the risk hierarchy)**	**Residual** Likelihood	Impact	Score	**Further controls (use the risk hierarchy)**
Contact with cutting tools/Sharp objects eg. Band saw.	Cuts, scrapes, abrasions	User	2	2	4	Ensure proper storage and guarding where applicable Ensure adequate knowledge of how to use tool/equipment	1	2	1	
Laser Cutter	Burns Blindness	User, and those in close Vicinity	1	3	3	Ensure safety glasses worn at all time when operating laser cutter. Only use as intended. Ensure adequate competency in operation of machine.	1	3	3	

1

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

PART A											
(1) Risk identification			**(2) Risk assessment**				**(3) Risk management**				
Hazard	**Potential Consequences**	**Who might be harmed** (user; those nearby; those in the vicinity; members of the public)	**Inherent** Likelihood	Impact	Score	**Control measures (use the risk hierarchy)**	**Residual** Likelihood	Impact	Score	**Further controls (use the risk hierarchy)**
Tools and equipment such as cutters and pliers	Pinches/cuts to hands and fingers	User	2	1	2	Ensure tools are in good working order Only use tools as intended	1	1	1	
Fumes From Laser cutter	Eye/breathing discomfort	User, and those in close Vicinity	2	1	2	Ensure use of fume extractor/filters. Avoid being too close to actively cutting machine.	1	1	1	
Use of adhesives	Skin contact Fumes from chemicals	User, and those in close Vicinity	2	2	4	Ensure well ventilated area prior to using volatile glues/adhesives Practice care when using	1	2	2	
Tripping and slipping	Bruising, Sprains, and potentially more serious injury	Any personnel in the area	1	3	3	Ensure work area is clear of debris, discarded materials, and is dry, spills cleared immediately.	1	3	3	

2

PART A										
(1) Risk identification			(2) Risk assessment				(3) Risk management			
Hazard	Potential Consequences	Who might be harmed (user; those nearby; those in the vicinity; members of the public)	Inherent			Control measures (use the risk hierarchy)	Residual			Further controls (use the risk hierarchy)
			Likelihood	Impact	Score		Likelihood	Impact	Score	

PART B – Action Plan

Risk Assessment Action Plan

Part no.	Action to be taken, incl. Cost	By whom	Target date	Review date	Outcome at review date
	-				

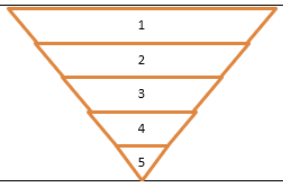
Responsible manager's signature: Print name: Date:				Responsible manager's signature: Print name: Date	

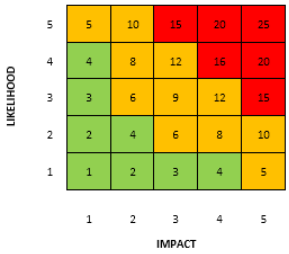
University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

Assessment Guidance

1. Eliminate	Remove the hazard wherever possible which negates the need for further controls	If this is not possible then explain why
2. Substitute	Replace the hazard with one less hazardous	If not possible then explain why
3. Physical controls	Examples: enclosure, fume cupboard, glove box	Likely to still require admin controls as well
4. Admin controls	Examples: training, supervision, signage	
5. Personal protection	Examples: respirators, safety specs, gloves	Last resort as it only protects the individual





Risk process

1. Identify the impact and likelihood using the tables above.
2. Identify the risk rating by multiplying the Impact by the likelihood using the coloured matrix.
3. If the risk is amber or red – identify control measures to reduce the risk to as low as is reasonably practicable.
4. If the residual risk is green, additional controls are not necessary.
5. If the residual risk is amber the activity can continue but you must identify and implement further controls to reduce the risk to as low as reasonably practicable.
6. If the residual risk is red do not continue with the activity until additional controls have been implemented and the risk is reduced.
7. Control measures should follow the risk hierarchy, where appropriate as per the pyramid above.
8. The cost of implementing control measures can be taken into account but should be proportional to the risk i.e. a control to reduce low risk may not need to be carried out if the cost is high but a control to manage high risk means that even at high cost the control would be necessary.

Impact		Health & Safety
1	Trivial - insignificant	Very minor injuries e.g. slight bruising
2	Minor	Injuries or illness e.g. small cut or abrasion which require basic first aid treatment even in self-administered.
3	Moderate	Injuries or illness e.g. strain or sprain requiring first aid or medical support.
4	Major	Injuries or illness e.g. broken bone requiring medical support >24 hours and time off work >4 weeks.
5	Severe – extremely significant	Fatality or multiple serious injuries or illness requiring hospital admission or significant time off work.

Likelihood	
1	Rare e.g. 1 in 100,000 chance or higher
2	Unlikely e.g. 1 in 10,000 chance or higher
3	Possible e.g. 1 in 1,000 chance or higher
4	Likely e.g. 1 in 100 chance or higher
5	Very Likely e.g. 1 in 10 chance or higher

5

A.4.1.3 Electronics workshop Method Statement

Faculty of Engineering and the Environment	**Method Statement**
Title **Electronics work for the building of a Bipedal Robot.**	
Location of Activity *Tizard Building 13, Room 3027*	**Date** **1/2/2022**
Assessor *Ryan Khoo Yeap Hong* *ryhk1a18 30480183*	**Contact Details** *ryhk1a18@soton.ac.uk* *+44 0749 376 4830*
Supervisor Dr Suleiman Sharkh	**Contact Details** *S.M.Sharkh@soton.ac.uk*

Introduction / Overview.

Background description to the project. What will you achieve? How will you do this? Why is this required?

A Bipedal robot is to be built, to do so, various electronic work will be done to enable the completion of the robot. This will be done in the Electronics workshop by utilising its tools and resources to: cut and strip wires, solder wires together, apply heat shrink to protect the wires. Plastic 3D-printed parts may also be assembled and modified, this may involve simple drilling, sanding, cutting and deburring.

Description of Task and how it will be carried out.

Including any diagrams, materials, samples and equipment to be used as applicable.

The main activity that will be carried out is splicing wires. This involves cutting the wires, stripping the insulation, twisting them together, soldering them together, and shrink wrapping a sleeve over the joint with a heat gun.

Control Measures including training, PPE

Identify significant hazards and actions/control measures to be taken.

Hazards present are soldering irons and heat guns, these tools will not be left turned on and unattended. They will also be stored properly when not in use (heat gun turned off, and soldering iron placed into its holder and no where else)

An official induction for the workshop was also attended to ensure adequate knowledge of how to utilise the various tools and equipment available.

Emergency Arrangements

-

Additional persons involved in activity

-

A.4.1.4 Electronics workshop Risk Assessment

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

Risk Assessment			
Risk Assessment for the activity of	**Electronic Works and related activities**	**Date**	**1/2/2022**
Unit/Faculty/Directorate	**Mechanical Engineering**	**Assessor**	**Ryan Khoo**
Line Manager/Supervisor	***Dr Suleiman Sharkh***	**Signed off**	

PART A											
(1) Risk identification			**(2) Risk assessment**				**(3) Risk management**				
Hazard	**Potential Consequences**	**Who might be harmed (user; those nearby; those in the vicinity; members of the public)**	**Inherent**			**Control measures (use the risk hierarchy)**	**Residual**			**Further controls (use the risk hierarchy)**
			Likelihood	**Impact**	**Score**		**Likelihood**	**Impact**	**Score**	
Contact with soldering irons	Burns, most likely on hands/fingers	User of soldering iron	2	1	2	**Ensure Iron is only placed in its holder when not in use Always assume Iron is hot Use of vice and pliers to manipulate workpieces close to Soldering iron**	1	1	1	
Fire caused by soldering Irons	Fires, Burns, destruction of property	User, all persons in immediate vicinity	1	3	3	**Ensure Area is neat Avoid placing flammables near work area Ensure Soldering iron is off and cool before leaving area**	1	3	3	

1

University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

PART A											
(1) Risk identification			**(2) Risk assessment**				**(3) Risk management**				
Hazard	**Potential Consequences**	**Who might be harmed (user; those nearby; those in the vicinity; members of the public)**	**Inherent**			**Control measures (use the risk hierarchy)**	**Residual**			**Further controls (use the risk hierarchy)**
			Likelihood	**Impact**	**Score**		**Likelihood**	**Impact**	**Score**	
Tools and equipment such as cutters and pliers	Pinches/cuts to hands and fingers	User	2	1	2	**Ensure tools are in good working order Only use tools as intended**	1	1	1	
Fumes From soldering	Eye/breathing discomfort	User, and those in close Vicinity	2	1	0	**Ensure good placement and use of fume extractor/filter.**	1	1	1	

2

PART A											
(1) Risk identification			(2) Risk assessment				(3) Risk management				
Hazard	Potential Consequences	Who might be harmed (user; those nearby; those in the vicinity; members of the public)	Inherent			Control measures (use the risk hierarchy)	Residual			Further controls (use the risk hierarchy)
			Likelihood	Impact	Score		Likelihood	Impact	Score	

PART B – Action Plan

Risk Assessment Action Plan

Part no.	Action to be taken, incl. Cost	By whom	Target date	Review date	Outcome at review date
	-				

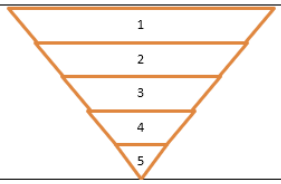
Responsible manager's signature: Print name: Date:	Responsible manager's signature: Print name: Date

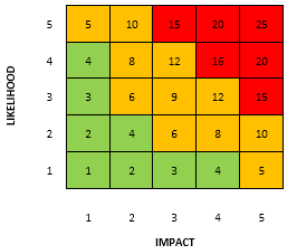
University of Southampton Health & Safety Risk Assessment

Version: 2.3/2017

Assessment Guidance

1. Eliminate	Remove the hazard wherever possible which negates the need for further controls	If this is not possible then explain why
2. Substitute	Replace the hazard with one less hazardous	If not possible then explain why
3. Physical controls	Examples: enclosure, fume cupboard, glove box	Likely to still require admin controls as well
4. Admin controls	Examples: training, supervision, signage	
5. Personal protection	Examples: respirators, safety specs, gloves	Last resort as it only protects the individual





Risk process

1. Identify the impact and likelihood using the tables above.
2. Identify the risk rating by multiplying the Impact by the likelihood using the coloured matrix.
3. If the risk is amber or red – identify control measures to reduce the risk to as low as is reasonably practicable.
4. If the residual risk is green, additional controls are not necessary.
5. If the residual risk is amber the activity can continue but you must identify and implement further controls to reduce the risk to as low as reasonably practicable.
6. If the residual risk is red do not continue with the activity until additional controls have been implemented and the risk is reduced.
7. Control measures should follow the risk hierarchy, where appropriate as per the pyramid above.
8. The cost of implementing control measures can be taken into account but should be proportional to the risk i.e. a control to reduce low risk may not need to be carried out if the cost is high but a control to manage high risk means that even at high cost the control would be necessary.

Impact		Health & Safety
1	Trivial - insignificant	Very minor injuries e.g. slight bruising
2	Minor	Injuries or illness e.g. small cut or abrasion which require basic first aid treatment even in self-administered.
3	Moderate	Injuries or illness e.g. strain or sprain requiring first aid or medical support.
4	Major	Injuries or illness e.g. broken bone requiring medical support >24 hours and time off work >4 weeks.
5	Severe – extremely significant	Fatality or multiple serious injuries or illness requiring hospital admission or significant time off work.

Likelihood	
1	Rare e.g. 1 in 100,000 chance or higher
2	Unlikely e.g. 1 in 10,000 chance or higher
3	Possible e.g. 1 in 1,000 chance or higher
4	Likely e.g. 1 in 100 chance or higher
5	Very Likely e.g. 1 in 10 chance or higher

5

A.5 Full Controller State Commands

State	State Number	Sub-States (magnitude)	Description
Joy N	10	10-19	Walk forward
Joy NE	20	20-29	Turn Right
Joy E	30	30-39	Stride right
Joy SE	40	40-49	
Joy S	50	50-59	Walk Backwards
Joy SW	60	60-69	
Joy W	70	70-79	Stride Left
Joy NW	80	80-89	Turn Left
Z button	90	-	
C Button	100	-	
C + Joy Pitch	110	110-119	Manual Spine Pitch Joy +-30 degrees
C + Joy Roll	120	120-129	Manual Spine roll Joy +-45 degrees
C+ Joy Left and right	130	130-139	Manual Spine Look Left and right

A.6 MATLAB Code

```
fuzzyLogicFile = Direction_Control ; %name of fuzzy logic file to be
utilised for output
outputSize = 51; %Size of output array, Fuzzy Logic surface will scale
accordingly
%output size should be an odd number to allow a middle number

arraySize = outputSize -1;
increment = 200/arraySize;
fileID = fopen('Fuzzy Logic Lookup Table Output.txt','w');

fprintf(fileID, "const int16_p PROGMEM fuzzTable [%.0f][%.0f] = { \n
",arraySize+1,arraySize+1);
for y = -100:increment:100
%print a row of variables
fprintf(fileID, "{");
for x = -100:increment:100
    fprintf(fileID, "%.0f", evalfis(fuzzyLogicFile,[x y])); %Round
number to closest whole number
    if x < 100
        fprintf(fileID, ", ");
    end
end
 fprintf(fileID, "}, \n  ");
end
fprintf(fileID, "}; \n");

buffer = extractFileText("Fuzzy Logic Lookup Table Output.txt");
disp(buffer);

fprintf("Center index: %f \n", arraySize/2);
```

A.7 Speed Test data

Speed tests	Time taken to travel 700mm (s)	Speed (mm/s)
Test 1	24	29.167
Test 2	21	33.333
Test 3	27	25.926
Test 4	22	31.818
Test 5	24	29.167
Test 6	25	28.000
Averaged	23.833	29.371

A.8 Arduino Code:

An Arduino library by (Madison 2021) was utilised to streamline the communication with the Nintendo “nunchuck” controller via an I2C interface.

[Accessed 29/4/2022] Available at: https://github.com/dmadison/NintendoExtensionCtrl

The in-built Arduino wire library was used to enable communication with “I2C” (Inter-Integrated Circuit) devices.

[Accessed 29/4/2022] Available at: https://www.arduino.cc/en/reference/wire

A library by Adafruit was utilised to interface with the servo driver.

[Accessed 29/4/2022] Available at: https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library

A library by Megunolink was utilised to simplify the implementation of the exponential filter.

[Accessed 29/4/2022] Available at: https://www.megunolink.com/documentation/arduino-library/

A library by Adafruit was utilised to interface with the IMU.

[Accessed 29/4/2022] Available at: https://github.com/adafruit/Adafruit_ICM20X

A library was utilised to streamline the implementation of storing and retrieving data from the program flash memory to free up SRAM.

[Accessed 29/4/2022] Available at: https://github.com/Chris--A/PGMWrap

A.8.1 Controller Code

```
//Ryan Khoo 2022, Bipedal Robot Controller Code

#include <NintendoExtensionCtrl.h>
Nunchuk nchuk;

int green = 8; //Red LED pin (0 for ON)
int red = 7; //Orange LED pin

void setup() {
  Serial.begin(9600);
  nchuk.begin();
  nchuk.connect();

  pinMode(red, OUTPUT);
```

```
  pinMode(green, OUTPUT);
  digitalWrite(green, 1);//green light starts off
  digitalWrite(red, 1);//red light starts off
}

void loop() {

  nchuk.update();
  //Obtain Current State of remote
  int Cbutton = nchuk.buttonC();
  int Zbutton = nchuk.buttonZ();
  int Xvalue = nchuk.joyX();//0-255, middle at 129
  int Yvalue = nchuk.joyY();//middle at 126
  int Roll = nchuk.rollAngle();
  int Pitch = nchuk.pitchAngle();
  //Serial.print("Yvalue:");Serial.println(Yvalue);



  //Manual Look control red light mode
  if (Cbutton == 1){
    digitalWrite(red, 0); //red light on
    //Manual Pitch
    int pitchCommand = map(Pitch,-30,30,110,119);
    if (pitchCommand >=110 && pitchCommand <= 119){
      Serial.write(pitchCommand);//-----------------------------------------
Command Send
    }
    //Manual Roll
    int rollCommand = map(Roll,-45,45,120,129);
    if (rollCommand >=120 && rollCommand <= 129){
      Serial.write(rollCommand);//-----------------------------------------
Command Send
    }
    //Manual Look left and right
    int lookCommand = map(Xvalue,0,253,130,139);
    if (lookCommand >=130 && lookCommand <= 139){
      Serial.write(lookCommand);//-----------------------------------------
Command Send
    }
  }

  //normal joy operation

    //green light when joy moved
    if(((Xvalue > 130 or Xvalue < 127)or((Yvalue > 127 or Yvalue < 125))) &&
Cbutton==0 && Zbutton ==0){
      digitalWrite(green, 0);//green light on

    //Forward
    int forwardCommand = map(Yvalue,129,255,10,19);
    if (forwardCommand >12 && forwardCommand <= 19){
      Serial.write(forwardCommand);//---------------------------------------
--Command Send
      //Serial.println(forwardCommand);

    }
        //Back
    int backCommand = map(Yvalue,0,126,59,50);
    if (backCommand >52 && backCommand <= 59){
```

```
      Serial.write(backCommand);//-----------------------------------------
Command Send
      //Serial.println(backCommand);

    }
        //Right
    int rightCommand = map(Xvalue,126,254,30,39);
    if (rightCommand >32 && rightCommand <= 39){
      Serial.write(rightCommand);//-----------------------------------------
Command Send
      //Serial.println(rightCommand);

    }

        //Left
    int leftCommand = map(Xvalue,0,125,79,70);
    if (leftCommand >72 && leftCommand <= 79){
      Serial.write(leftCommand);//-----------------------------------------
Command Send
      //Serial.println(leftCommand);

    }
    delay(100);



    }

  //temp z button activation
  if (Zbutton == 1){
    digitalWrite(green, 0);//yellow light on
    digitalWrite(red, 0);
  }



  //Yellow for Z button and green for normal moving operation



//Turn off lights if nothing pressed
  if (Yvalue==126 && (Xvalue==129 or Xvalue==128) && Zbutton==0 && Cbutton ==
0){
    digitalWrite(green, 1);//green light off
    digitalWrite(red, 1);//red light off
  }
  delay(30); //delay to avoid overloading bluetooth bus
}
```

A.8.2 Robot Code

```
// Ryan Khoo Yeap Hong | 30480183 | Year 3 Individual Project |
Academic year 2021/2022 | ryankhoo@ymail.com | ryhk1a18@soton.ac.uk
#include <Wire.h>
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
#include <Adafruit_ICM20948.h>
Adafruit_ICM20948 icm;
Adafruit_Sensor *icm_gyro, *icm_mag, *icm_accel;
#include <PGMWrap.h>
#include "Filter.h"

//Changeable Parameters
0000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000

//Walking pattern progress tracked by:
int cycleStage ;
int mode = 9; //<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--
<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<--<-
-
// 0 to disable all cycles (for debugging)
// 9 to begin in dynamic neutral
// 1 for normal operation
// 10 to enable servo initialisations
// 20 for neutral position
// 30 for custom test position

//Servo execution speed (in miliseconds between each state execution,
smaller number = faster execution)
int setDelayX = 2; //Default = 2, the time between each "eating" of a
pending set, minimum depends on how fast void loop can execute
int setDelay = 2;// Code usues this variable, but "X" version is to
allow initialisation to go slower before main loop

//Delay between IMU measurements
int IMUDelay = 20;

// footstep set modifiers
int emptyMod = 40; // Default = 40, 0 to 100, adjusts height of empty
step, mostly unchanged.
int xMod = 0; //-100 to 100, degree of forward stepping 100 = front
full, set by code
int yMod = 0; //-100 to 100, degree of side stepping 100 = right full,
set by code

const int gaitAngle = 10; // Default = 20, the angle that the legs will
swing forward and back, smaller angles require less torque, 50 for demo
const int gaitSideAngle = -10; // Default = 20, the angle that the legs
will Left and Right, smaller angles require less torque, 50 for demo

//max divisions possible, may need to be reduce depending on memory
use, the higher the better/smoother.
const int maxDivisions = 10;
```

```
//when both X and Y modifiers are below stillThreshold, robot will
stand still
int stillThreshold = 20;

//Modifier Weightages (100 = 1)
int fuzzWeight = 100;
int controllerWeight = 100;

//Filter Weights
ExponentialFilter<float > filteredRoll(5, 0);
ExponentialFilter<float> filteredPitch(5, 0);
ExponentialFilter<float > filteredAngVelX(5, 0);
ExponentialFilter<float > filteredAngVelY(8, 0);

//0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000
//1111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111
//Movement sets
//Time Variable may be added in the tenth slot [9], values are in
divisions. i.e. a value of 10 would mean 10 divisions between the
previous state and the new input set,
//more divisions = slower, time to assume new state in microseconds: t
= divisions * setDelay

//Initialisation sets ------------------------------------------------
-----------------------------
const int16_p PROGMEM startServoState[]   = {0,0,0,0,0,0,0,0,0,1}; //
Intant recenter, To neutral position at max speed CURRENTLY UNUSED
const int16_p PROGMEM neutralServoState[] = {0,0,0,0,0,0,0,0,0,10};//
Servos centered, with last variable the time variable
const int16_p PROGMEM maxServoState[]     =
{100,60,100,100,60,100,0,0,0,10};// Servos at full maximums
const int16_p PROGMEM minServoState[]     = {-100,-100,-50,-100,-100,-
50,0,0,0,10};// Servos at full miniimums
//Test Leg state
const int16_p PROGMEM testServoState[] = {0,0,100,0,0,100,0,0,0,6};//
ONLY for testing

//Empty walking sets, defines time variable, other sets time value = 0
------------------------------------------------------------------------
-----------
// int emptyMod = 100 This is above in changable states, adjusts height
of step
//Only empty sets have a time value to differentiate durations of
various stages
//                                          L    L   L   R    R   R   M
M   M   t
const int16_p PROGMEM empty1ServoState[] = {0,   0,  0, -50, -40, 0, -
20, 0, 20,  4}; // retracted right leg
const int16_p PROGMEM empty2ServoState[] = {0,   0,  0, -50, -40, 0, -
10, 0, 20,  4};
```

```
const int16_p PROGMEM empty3ServoState[] = {0,   0,  0,  0,   0,  0,
0,  0,  0,  6};
const int16_p PROGMEM empty4ServoState[] = {0,   0,  0,  0,   0,  0,
0,  0,  0,  6};
const int16_p PROGMEM empty5ServoState[] = {-50,-40, 0,  0,   0,  0,
20,  0, -20, 4}; //retracted left leg
const int16_p PROGMEM empty6ServoState[] = {-50,-40, 0,  0,   0,  0,
10,  0, -20, 4};
const int16_p PROGMEM empty7ServoState[] = {0,   0,  0,  0,   0,  0,
0,  0,  0,  6};
const int16_p PROGMEM empty8ServoState[] = {0,   0,  0,  0,   0,  0,
0,  0,  0,  6};
//Front-back walking sets ------------------------------------------
----------------------------------------------------------------------
----------
const int GD = gaitAngle/10; //GD = Gait division
const int16_p PROGMEM x1ServoState[] = {0,GD*-1,0,0,GD*5,0,0,0,0,0};
const int16_p PROGMEM x2ServoState[] = {0, GD*1,0,0,GD*-5,0,60,0,0,0};
const int16_p PROGMEM x3ServoState[] = {0, GD*3,0,0,GD*-5,0,30,0,0,0};
const int16_p PROGMEM x4ServoState[] = {0, GD*5,0,0,GD*-3,0,0,0,0,0};
const int16_p PROGMEM x5ServoState[] = {0, GD*5,0,0,GD*-1,0,0,0,0,0};
const int16_p PROGMEM x6ServoState[] = {0,GD*-5,0,0,GD*1,0,-60,0,0,0};
const int16_p PROGMEM x7ServoState[] = {0,GD*-5,0,0,GD*3,0,-30,0,0,0};
const int16_p PROGMEM x8ServoState[] = {0,GD*-3,0,0,GD*5,0,0,0,0,0};
//Left-right walking sets ------------------------------------------
----------------------------------------------------------------------
----------

const int GSD = gaitSideAngle/10; //GD = Gait side division, simmilar
to above
const int16_p PROGMEM y1ServoState[] = {0,0,GSD*-1,0,0,GSD*-5,0,0,0,0};
const int16_p PROGMEM y2ServoState[] = {0,0,GSD*1,0,0,GSD*5,0,0,0,0};
const int16_p PROGMEM y3ServoState[] = {0,0,GSD*3,0,0,GSD*5,0,0,0,0};
const int16_p PROGMEM y4ServoState[] = {0,0,GSD*5,0,0,GSD*3,0,0,0,0};
const int16_p PROGMEM y5ServoState[] = {0,0,GSD*5,0,0,GSD*1,0,0,0,0};
const int16_p PROGMEM y6ServoState[] = {0,0,GSD*-5,0,0,GSD*-1,0,0,0,0};
const int16_p PROGMEM y7ServoState[] = {0,0,GSD*-5,0,0,GSD*-3,0,0,0,0};
const int16_p PROGMEM y8ServoState[] = {0,0,GSD*-3,0,0,GSD*-5,0,0,0,0};
// ----------------------------------------------------------------------
----------------------------------------------------------------------
----------
//11111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111
11111111111111111
// Constant parameters and various other variable definitions
22222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222
222222222222222222

//Servo Numbers ----------------------------------------------------
----------------------------------------------------------------------
-------------------------------------
const int16_p PROGMEM  LKneeServo = 0;
const int16_p PROGMEM  LHipServo = 1;
const int16_p PROGMEM  LHipSideServo = 2;
```

```
const int16_p PROGMEM  RKneeServo = 15;
const int16_p PROGMEM  RHipServo = 14;
const int16_p PROGMEM  RHipSideServo = 13;
const int16_p PROGMEM  lookServo = 4;
const int16_p PROGMEM  pitchServo = 5;
const int16_p PROGMEM  rollServo = 6;
//the above in a set for utilisations in fucntions:
const int servoSet[] = {LKneeServo, LHipServo, LHipSideServo,
RKneeServo, RHipServo, RHipSideServo, lookServo, pitchServo,
rollServo};

//Calibration for centers of servos, i.e. neutral angular positions,  -
------------------------------------------------------------------------
---------------------------------------
const int16_p PROGMEM  LKneeCenter = 100; //clockwise looking left
const int16_p PROGMEM  LHipCenter = 88+2;
const int16_p PROGMEM  LHipSideCenter = 90; //lowet, inner leg
const int16_p PROGMEM  RKneeCenter = 165; //counterclockwise looking
left
const int16_p PROGMEM  RHipCenter = 90-2;
const int16_p PROGMEM  RHipSideCenter = 85;//Higher, leg goes towards
center line
const int16_p PROGMEM  lookCenter = 93;//lower look right
const int16_p PROGMEM  pitchCenter = 88;//lower for higher head
const int16_p PROGMEM  rollCenter = 95;
//the above in a set for utilisations in fucntions:
const int servoCenter[] = { LKneeCenter, LHipCenter, LHipSideCenter,
RKneeCenter, RHipCenter, RHipSideCenter, lookCenter, pitchCenter,
rollCenter};// Servo centers in a set

//Movement ranges, maximum degrees of movement for each joint ---------
------------------------------------------------------------------------
-------------------------------------
const int16_p PROGMEM  LKneeRange = 50;//range of degrees of Left Knee
servo is +-60
const int16_p PROGMEM  LHipRange = 50;
const int16_p PROGMEM  LHipSideRange = 40;
const int16_p PROGMEM  RKneeRange = -50;
const int16_p PROGMEM  RHipRange = -50;
const int16_p PROGMEM  RHipSideRange = -40;
const int16_p PROGMEM  lookRange = 30;
const int16_p PROGMEM  pitchRange = 30;
const int16_p PROGMEM  rollRange = 35;
//the above in a set for utilisations in fucntions:
const int servoRange[] = {LKneeRange, LHipRange, LHipSideRange,
RKneeRange, RHipRange, RHipSideRange, lookRange, pitchRange,
rollRange};// Servo ranges in a set

//initial command state sent over bluetooth
int state = 0;

//needed for delayed set execution loop below
int timeOfLastLoop = 0;
//needed for delayed IMU measurement loop below
int timeOfLastIMU = 0;
```

```
float accAngleX, accAngleY, gyroAngleX, gyroAngleY, gyroAngleZ, yaw,
roll, pitch;
  float GyroX;
  float GyroY;
//needed for delayed ultrasonic measurement loop below, in microseconds
int updateDistance = 0; //is set to 1 when distance needs to be
updated.
int timeOfLastUltra = 0;
int ultraState = 0; //controls if the ultrasonic sensor is resting or
pulsing for 10 microseconds.
int distanceSensed; //global variable storing distance of obastacle
sensend by ultrasonic sensor, updates every 20 microseconds.

//temporary memory of its current position, for calculation of eased
servo movement to the new state
int oldSet[] = {0,0,0,0,0,0,0,0,0,1}; //old set starts with neutral
positions

//the "plate" of sets the code will need to "eat" one by one in the
loop every setDelay amount of time.
int pendingSet[maxDivisions+1][9];
  //pendingSet[maxDivisions][9] = 0; // This line is in setup, the
maxdivisions value in the set states how many "edible" sets are
currently being stored and can be "eaten"

//Set that generate step writes to
int generatedSet[10];

//variable to allow neutral position to be executed only once upon
entering neutral position
int enterNeutral = 1;

//Lookup table for fuzzy logic
const int16_p PROGMEM fuzzTable[51][51] = {
 {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34,
-34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -17, -
17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
```

```
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -17, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -34, -
34, -34, -33, -32, -31, -30, -28, -27, -26, -24, -23, -21, -20, -18, -
17, -16, -16, -15, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-33, -33, -33, -33, -33, -33, -33, -33, -33, -33, -33, -33, -33, -
33, -33, -33, -32, -31, -30, -28, -27, -25, -24, -23, -21, -20, -18, -
17, -16, -16, -15, -15, -14, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -14, -14},
  {-31, -31, -31, -31, -31, -31, -31, -31, -31, -31, -31, -31, -31, -
31, -31, -31, -31, -30, -29, -28, -26, -25, -23, -22, -21, -19, -18, -
17, -16, -15, -15, -14, -14, -14, -14, -14, -13, -13, -13, -13, -13, -
13, -13, -13, -13, -13, -13, -13, -13, -13, -13},
  {-29, -29, -29, -29, -29, -29, -29, -29, -29, -29, -29, -29, -29, -
29, -29, -29, -29, -29, -28, -27, -25, -24, -22, -21, -19, -18, -17, -
16, -15, -14, -14, -13, -13, -13, -13, -13, -12, -12, -12, -12, -12, -
12, -12, -12, -12, -12, -12, -12, -12, -12, -12},
  {-26, -26, -26, -26, -26, -26, -26, -26, -26, -26, -26, -26, -26, -
26, -26, -26, -26, -26, -26, -25, -24, -22, -21, -19, -18, -17, -15, -
14, -14, -13, -12, -12, -12, -12, -11, -11, -11, -11, -11, -11, -11, -
11, -11, -11, -11, -11, -11, -11, -11, -11, -11},
  {-23, -23, -23, -23, -23, -23, -23, -23, -23, -23, -23, -23, -23, -
23, -23, -23, -23, -23, -23, -23, -22, -21, -19, -18, -16, -15, -14, -
```

```
13, -12, -11, -11, -11, -10, -10, -10, -10, -10, -10, -10, -10, -
10, -10, -10, -10, -10, -10, -10, -10, -10},
  {-20, -20, -20, -20, -20, -20, -20, -20, -20, -20, -20, -20, -20, -
20, -20, -20, -20, -20, -20, -20, -20, -19, -17, -16, -15, -13, -12, -
11, -10, -10, -9, -9, -9, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -8, -
8, -8, -8, -8, -8, -8},
  {-17, -17, -17, -17, -17, -17, -17, -17, -17, -17, -17, -17, -17, -
17, -17, -17, -17, -17, -17, -17, -17, -15, -14, -13, -11, -10, -
9, -8, -8, -7, -7, -7, -7, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6, -6,
-6, -6, -6, -6, -6, -6},
  {-14, -14, -14, -14, -14, -14, -14, -14, -14, -14, -14, -14, -14, -
14, -14, -14, -14, -14, -14, -14, -14, -13, -13, -12, -10, -9, -8, -7,
-6, -5, -5, -5, -5, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,
-4, -4, -4, -4, -4, -4},
  {-11, -11, -11, -11, -11, -11, -11, -11, -11, -11, -11, -11, -11, -
11, -11, -11, -11, -10, -10, -10, -10, -10, -10, -9, -8, -6, -5, -4, -
4, -3, -3, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2,
-2, -2, -2, -2, -2, -2},
  {-7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7, -7,
-7, -7, -7, -7, -7, -6, -5, -5, -3, -2, -1, -1, -0, 0, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
  {-4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4, -4,
-4, -4, -4, -4, -3, -3, -3, -2, -1, -0, 1, 2, 3, 3, 3, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4},
  {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -0, -0, 0, 1, 1, 2, 3, 5, 5, 6, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7,
7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7, 7},
  {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 4,
4, 5, 6, 8, 9, 10, 10, 10, 10, 10, 10, 11, 11, 11, 11, 11, 11, 11, 11,
11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11},
  {4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 6,
7, 8, 9, 10, 12, 13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14},
  {6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8,
9, 10, 11, 13, 14, 15, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17, 17},
  {8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 10,
10, 11, 12, 13, 15, 16, 17, 19, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20,
20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20},
  {10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10,
10, 10, 10, 11, 11, 11, 12, 13, 14, 15, 16, 18, 19, 21, 22, 23, 23, 23, 23,
23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23, 23},
  {11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11, 11,
12, 12, 12, 12, 13, 14, 14, 15, 17, 18, 19, 21, 22, 24, 25, 26, 26, 26,
26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26, 26},
  {12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 12, 13, 13,
13, 13, 13, 14, 14, 15, 16, 17, 18, 19, 21, 22, 24, 25, 27, 28, 29, 29,
29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29, 29},
  {13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 14, 14,
14, 14, 14, 15, 15, 16, 17, 18, 19, 21, 22, 23, 25, 26, 28, 29, 30, 31,
31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31, 31},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 14, 14, 15, 15, 16, 16, 17, 18, 20, 21, 23, 24, 25, 27, 28, 30, 31, 32,
33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33, 33},
```

```
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 16, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  {14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14,
14, 15, 15, 15, 16, 17, 17, 18, 20, 21, 23, 24, 26, 27, 28, 30, 31, 32,
33, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34, 34},
  };

//2222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222
2222222222
// Predefined Functions and calculations
33333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333
3333333333333333333333333
```

```
//Servodriver Pulse calculations -------------------------------------
-------------------------------------------------------------------
-----------------
int anglePulse (int angle){
  int Pulse = map(angle, 0, 180, 110, 400);
  return Pulse;
//, MIN pulse = 110, Max pulse = 610, pulse to 180*= 400
}

//executePosition function, Write position dataset to servos ----------
-------------------------------------------------------------------
-----------------------------------
void executePosition( int pendingSetIndex ){
  //Serial.println(F("executed positions"));
  for (int i=0; i<9; i++){
    int writeDegree = map(pendingSet[pendingSetIndex][i], -100, 100, (-
servoRange[i]+servoCenter[i]), (servoRange[i]+servoCenter[i]) ); //maps
the -100 to 100 input to the maximum and minimums of each servo
    pwm.setPWM(servoSet[i], 0, anglePulse(writeDegree));
    //Serial.println(pendingSet[pendingSetIndex][i]);

  }
}

//Assume position function -------------------------------------------
-------------------------------------------------------------------
-----------------------------------
//generates a new array of arrays into pendingSet that the loop will
work through, Nature determines how the movement is performed, either
Linear (0) or sine-eased (1)
void assumePosition(int newSet[], int nature){
  //One servo number at a time
//Serial.println("assumePosition() started");

//set of values to be added to pending sets once generated
  int transitionArray[9][maxDivisions] ;
  int divisions = newSet[9]; //the 9th value in a servo set is the time
value

  for (int servoNumber = 0; servoNumber<9;servoNumber++){

    //First, Multiplier array is made depending on the nature selected
    float multiplierArray[maxDivisions];
    int oldValue = oldSet[servoNumber];
    int newValue = newSet[servoNumber];
    float increment = 0;

    //Generate Multiplier Array
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    if (nature == 0){//Linear movement, constant speed
      increment = ( 1.0 / divisions);

      for ( int i = 0; i < divisions ; i++){
        float multiplier = increment * (i+1);
        multiplierArray[i] = multiplier;
      }
```

```
    }
    if (nature == 1){//Sin movement, deccelerate
      increment = ( 1.570796 / divisions);

      for ( int i = 0; i < divisions ; i++){
        float multiplier = sin(increment * (i+1) );
        multiplierArray[i] = multiplier;
      }
    }
    if (nature == 2){//Cos movement, accelerate
      increment = ( 1.570796 / divisions);

      for ( int i = 0; i < divisions ; i++){
        float multiplier = (cos((increment * (i+1))+ 3.1415926 )) + 1;
        multiplierArray[i] = multiplier;
      }
    } // Multiplier array now generated
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    //make column of values for current servo it is calculating
    for ( int i = 0; i < divisions ; i++){

      int value = oldValue + ((newValue-oldValue)*multiplierArray[i]);
      transitionArray[servoNumber][i] = value;
      }
  }// end of respective servo number loop


//write to pending sets from transition array, transposing matrix
  for ( int i = 0; i < divisions ; i++){
    for ( int s = 0; s < 9 ; s++){
      pendingSet[divisions-i-1][s] = transitionArray[s][i];
      //Serial.println(pendingSet[divisions-i-1][s]);
    }
    //Serial.print("one pending set added at index: ");
    //Serial.println(divisions-i-1);
  }
pendingSet[maxDivisions][8] = divisions;//tell pending set how many
things there are to "eat"

//make oldSet = newSet
  for (int i=0; i<9; i++){
    oldSet[i] = newSet[i];
  }

} // end of assume position function

//Step Direction Combining ------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
------
//takes all 3 sets and their modifiers/weightages and updates
"generatedSet" to be the new set to be written
void generateSet( int set1[] , int set1Mod , int set2[] , int set2Mod
,int set3[] , int set3Mod ){
  // 10 values to combine and output, modifiers are 0-100,
```

```
  for (int i=0; i<9; i++){
    int newValue = (set1[i]*set1Mod) + (set2[i]*set2Mod) +
(set3[i]*set3Mod);
    generatedSet[i] = newValue / 100;
  }
  generatedSet[9] = set1[9];

//    for (int i=0; i<10; i++){
//    Serial.print(outputSet[i]);
//    Serial.print(", ");
//  }
//  Serial.println(" ");

}


//333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333333333333333333
333333333333333333333333333333333333333333333333333333333333333333
333333333333
//444444444444444444444444444444444444444444444444444444444444444444
444444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444


//Setup
void setup() {
  Serial.println(F("Void Setup Begin."));
  Serial.begin(9600);

  pwm.begin();//ServoDriver Required
  pwm.setPWMFreq(60);  // This is the maximum PWM frequency
  yield();

  //Wait for IMU to initialise
  if (!icm.begin_I2C()) {
    Serial.println(F("Failed to locate ICM20948 chip"));
    while (1) {
      delay(10);
    }
  }
  icm_gyro = icm.getGyroSensor(); //Initialise Gyro
  icm_mag = icm.getMagnetometerSensor(); //Initialise Magnetometer
  icm_accel = icm.getAccelerometerSensor();

  pinMode(10, OUTPUT); //ultrasonic sensor pin initialisation
  pinMode(11, INPUT);


  int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = neutralServoState[i]; }
   assumePosition(bufferSet, 1);
   cycleStage = mode; // Primes cycle stage to begin, starts with
dynamic neutral stage 9
```

```
   delay(3000); //3 seconds to place robot on surface before IMU takes
calibration data of "flat surface"


} //end of setup

//4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444444444444444444
44444444444
//5555555555555555555555555555555555555555555555555555555555555555
5555555555555555555555555555555555555555555555555555555555555555
555555555555555555555555555555555555555555555555555
// Void Loop
void loop() {

//Serial.print("cycleStage: "); Serial.print(cycleStage);
Serial.print(", xMod:"); Serial.print(xMod); Serial.print(", yMod:");
Serial.println(yMod);

  //Serial.println();
  //check controller command data
  //Read Command
 if(Serial.available() > 0){ // Checks whether data is coming from the
serial port
    state = Serial.read(); // Reads the data from the serial port
    //Serial.print("Bluetooth: "); Serial.println(state);
 }

    int currentTime = millis();
  //int timeOfLastIMU = 0; this line is in section 222222

  //Start of ultrasonic sensor ------------------------------------
-------------------------
  if (updateDistance == 1){
  int microsecs = micros(); //micros utilised to avoid using
delayMicroseconds()
  int timeSinceLastUltra = microsecs - timeOfLastUltra;
  if (timeSinceLastUltra >= 10){ //this if function runs every 10
microseconds
    if (ultraState == 0){ //turn on pulse
      digitalWrite(10, HIGH);
      ultraState = 1;
    }
    if (ultraState == 1){ //turns off pulse and records duration,
returns distance
      digitalWrite(10, LOW );
      int duration = pulseIn(11, HIGH);
      distanceSensed = duration * 0.034 / 2;
      //Serial.println(distanceSensed);
      ultraState = 0;
    }
  }
  updateDistance = 0;
  } //End of ultrasonic sensor ------------------------------------
-------------------------
```

```
  //Start of IMU
**************************************************************************
**************************************************************************
*************************
  int timeSinceLastIMU = currentTime - timeOfLastIMU;
  if (timeSinceLastIMU > IMUDelay - 1 ){ // Start of delayed IMU loop,
be=ased on IMU delay
  //Obtain updated IMU readings
  sensors_event_t gyro;
  sensors_event_t mag;
  sensors_event_t accel;
  icm_gyro->getEvent(&gyro);
  icm_mag->getEvent(&mag);
  icm_accel->getEvent(&accel);

  float AccX = accel.acceleration.x;
  float AccY = accel.acceleration.y;
  float AccZ = accel.acceleration.z;
  float MagX = mag.magnetic.x;
  float MagY = mag.magnetic.y;
  float MagZ = mag.magnetic.z;
  GyroX = (gyro.gyro.x *57.3) +1.45;// convert to Deg/s, account for
error. 1.45, these 2 are declared above
  GyroY = (gyro.gyro.y *57.3) -0.19;
  float GyroZ = (gyro.gyro.z *57.3) ;
  // float roll, pitch; in variables

  //Calculate position from acceleraometer data
  accAngleX = (atan(AccY / sqrt(pow(AccX, 2) + pow(AccZ, 2))) * 180 /
PI) +2.4; //error of 2.4
  accAngleY = (atan(-1 * AccX / sqrt(pow(AccY, 2) + pow(AccZ, 2))) *
180 / PI) -2.4;

  //Calculate position from Gyro data
  float elapsedTime = (timeSinceLastIMU / 1000.0);

  gyroAngleX = gyroAngleX + GyroX * elapsedTime; // deg/s * s = deg
  gyroAngleY = gyroAngleY + GyroY * elapsedTime;
  yaw =  yaw + GyroZ * elapsedTime;

  //Combine data for more accurate positiining w/out drift
  pitch = 0.96 * gyroAngleX + 0.04 * accAngleX;
  roll = 0.96 * gyroAngleY + 0.04 * accAngleY;

  //Serial print X and Y positioning
  //Serial.print(pitch);Serial.print(", ");
//  Serial.print(roll);
//  Serial.println();

    timeOfLastIMU = currentTime;
  }//end of delayed IMU loop ---------------------------------------

//end of IMU
**************************************************************************
```

```
**************************************************************************
************************

//Produce Direction Information, Xmod and Ymod based on 3 inputs:
ultrasonic sensor data, IMU/fuzzy logic, and control info
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

//First, map IMU outputs to fuzzy logic Crisp inputs.
//Filter IMU data to avoid spikes/bumpiness from walking
filteredPitch.Filter(pitch); //angle varies +-20
filteredRoll.Filter(roll); //angle varies +-20
filteredAngVelX.Filter(GyroX); // +-100
filteredAngVelY.Filter(GyroY); // +-100

//Map extremes of value into fuzzy logic crisp input table, update
depending on size of lookup table
int crispAngX = map(filteredPitch.Current() , -20,20,0,51);
int crispAngY = map(filteredRoll.Current() , -20,20,0,51);
int crispAngVelX = map(filteredAngVelX.Current() , -100,100,0,51);
int crispAngVelY = map(filteredAngVelY.Current() , -100,100,0,51);

//  Serial.print(GyroY);Serial.print(", ");
//  Serial.print(filteredAngVelY.Current());
//  Serial.println();

//extract data from look-up table to get fuzzXmod and fuzzYmod
int fuzzXmod;
int fuzzYmod;
if(crispAngX < 50 && crispAngX > 0 && crispAngY < 50 && crispAngY > 0
&& crispAngVelX < 50 && crispAngVelX > 0 && crispAngVelY < 50 &&
crispAngVelY > 0 ){ //if statment to ensure values are within the
bounds of lookuptable and does not extract random data from ram
fuzzXmod = fuzzTable[crispAngX][crispAngVelX];
fuzzYmod = fuzzTable[crispAngY][crispAngVelY];
}

//extract data from ultrasonic sensor
int obstacleMod;
if (distanceSensed < 20){
  obstacleMod = -20 ; //if less than 20cm, nudge backwards
}
if (distanceSensed >= 20){
  obstacleMod = 0 ;
}

//extract data from remote controller
int controllerModX = 0;
int controllerModY = 0;
int controllerModXA = 0;
int controllerModXB = 0;
int controllerModYA = 0;
int controllerModYB = 0;
if (state >= 10 && state <= 19){ //front
  int frontState = state - 10;
    controllerModXA = frontState*3; //adds a maximum nudge of 30 to mod
 }
```

```
if (state >= 50 && state <= 59){ //back
  int backState = state - 50;
    controllerModXB = -backState*3; //adds a maximum nudge of -30 to
mod
 }
if (state >= 30 && state <= 39){ //right
  int rightState = state - 30;
    controllerModYA = rightState*3; //adds a maximum nudge of 30 to mod
 }
if (state >= 70 && state <= 79){ //left
  int leftState = state - 70;
    controllerModYB = -leftState*3; //adds a maximum nudge of -30 to
mod
 }
 controllerModX = controllerModXA + controllerModXB;
 controllerModY = controllerModYA + controllerModYB;



//Serial.print("ControllerModX:
");Serial.print(controllerModX);Serial.print(" ControllerModY:
");Serial.println(controllerModY);

//combine modifiers according to predetermined weights

xMod = (fuzzXmod*fuzzWeight*2 + controllerModX*controllerWeight +
obstacleMod*100)/100;
yMod = (fuzzYmod*fuzzWeight*2 + controllerModY*controllerWeight)/100;

//Xmod and Ymod found
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


// Stand at neutral:
  if (cycleStage == 20 && pendingSet[maxDivisions][8] == 0){
    if (enterNeutral == 1){
    //extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = startServoState[i]; }
    assumePosition(bufferSet, 0);
    enterNeutral = 0;
    }
  }
// Test State position:
  if (cycleStage == 30 && pendingSet[maxDivisions][8] == 0){

    //extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = testServoState[i]; }

    assumePosition(bufferSet, 0);
  }
```

```
////first time initialisation loop ------------------------------------
---------------------------------
//int cycleStage == 10 this is in changeable variables
  if (cycleStage == 10 && pendingSet[maxDivisions][8] == 0){
    //extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = minServoState[i]; }
    assumePosition(bufferSet, 2);
    cycleStage = 11;
  }
  if (cycleStage == 11 && pendingSet[maxDivisions][8] == 0){
//extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = maxServoState[i]; }
    assumePosition(bufferSet, 0);
    cycleStage = 12;
  }
  if (cycleStage == 12 && pendingSet[maxDivisions][8] == 0){
//extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = neutralServoState[i]; }
    assumePosition(bufferSet, 1);
    cycleStage = 1; // Primes cycle stage to begin
  }
////end of first time initialisation ----------------------------------
---------------------------------


//55555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555555555555555555555555555555555555
555
//66666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666666666666666666666666666666
66


//CYCLE STAGES
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++


//Dynamic Neutral, stands still when modifiers are low

if (cycleStage == 9  && (not(xMod < stillThreshold && xMod > -
stillThreshold ) || not(yMod < stillThreshold && yMod > -
stillThreshold)) && pendingSet[maxDivisions][8] == 0) { //break out of
neutral (stage 9)
  cycleStage = 2; //moves rightfoot first
  enterNeutral = 1;
}
```

```
if ( (xMod < stillThreshold && xMod > -stillThreshold ) && (yMod <
stillThreshold && yMod > -stillThreshold) &&
pendingSet[maxDivisions][8] == 0){ //forced into neutral
  cycleStage = 9;
  updateDistance = 1; //take distance measurements when stationary

  //only write neutral positions once upon entering neutral
  //int enterNeutral = 0; //this is in pre setup
  if (enterNeutral == 1){
//extract set from PROGMEM into buffer set
    int bufferSet[10];
  for (int i=0; i<10; i++){ bufferSet[i] = neutralServoState[i]; }
    assumePosition(bufferSet, 1);
    enterNeutral = 0; // prevents constant writing of neutral position
during rest
  }
  }


//Normal cycle stages
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++

  if (cycleStage == 1 && pendingSet[maxDivisions][8] == 0){
    updateDistance = 1; //enables the ultrasonic sensor to obtain a
distance reading.
    setDelay = setDelayX; //makes setdelayX the chosen variable above
instead of slower initialisation variable.
    state = 0;// reset bluetooth data state

 //Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty1ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x1ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y1ServoState[i]; }

    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 2;
  }
  if (cycleStage == 2 && pendingSet[maxDivisions][8] == 0){
  //Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty2ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x2ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y2ServoState[i]; }
```

```
    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 3;
  }
  if (cycleStage == 3 && pendingSet[maxDivisions][8] == 0){
 //Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty3ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x3ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y3ServoState[i]; }
    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 4;
  }
  if (cycleStage == 4 && pendingSet[maxDivisions][8] == 0){
    state = 0;// reset bluetooth data state
//Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty4ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x4ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y4ServoState[i]; }

    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 5;
  }
  if (cycleStage == 5 && pendingSet[maxDivisions][8] == 0){
//Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty5ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x5ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y5ServoState[i]; }

    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 6;
  }
if (cycleStage == 6 && pendingSet[maxDivisions][8] == 0){
//Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
```

```
  for (int i=0; i<10; i++){ bufferSetA[i] = empty6ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x6ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y6ServoState[i]; }
    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 7;
  }
  if (cycleStage == 7 && pendingSet[maxDivisions][8] == 0){
//Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty7ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x7ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y7ServoState[i]; }

    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 8;
  }

    if (cycleStage == 8 && pendingSet[maxDivisions][8] == 0){
   //Extract sets from PROGMEM into SRAM buffersets
    int bufferSetA[10];
  for (int i=0; i<10; i++){ bufferSetA[i] = empty8ServoState[i]; }

    int bufferSetB[10];
  for (int i=0; i<10; i++){ bufferSetB[i] = x8ServoState[i]; }

    int bufferSetC[10];
  for (int i=0; i<10; i++){ bufferSetC[i] = y8ServoState[i]; }
    generateSet(bufferSetA, emptyMod, bufferSetB, xMod, bufferSetC,
yMod );
    assumePosition(generatedSet, 0);
    cycleStage = 1;
  }
//END OF CYCLE STAGE
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//66666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666666666666666666666666666666
66666666666666666666666666666666666666666666666666
//77777777777777777777777777777777777777777777777777777777777777777
77777777777777777777777777777777777777777777777777777777777777777
777777777777777777777777777777777777777777777777
```

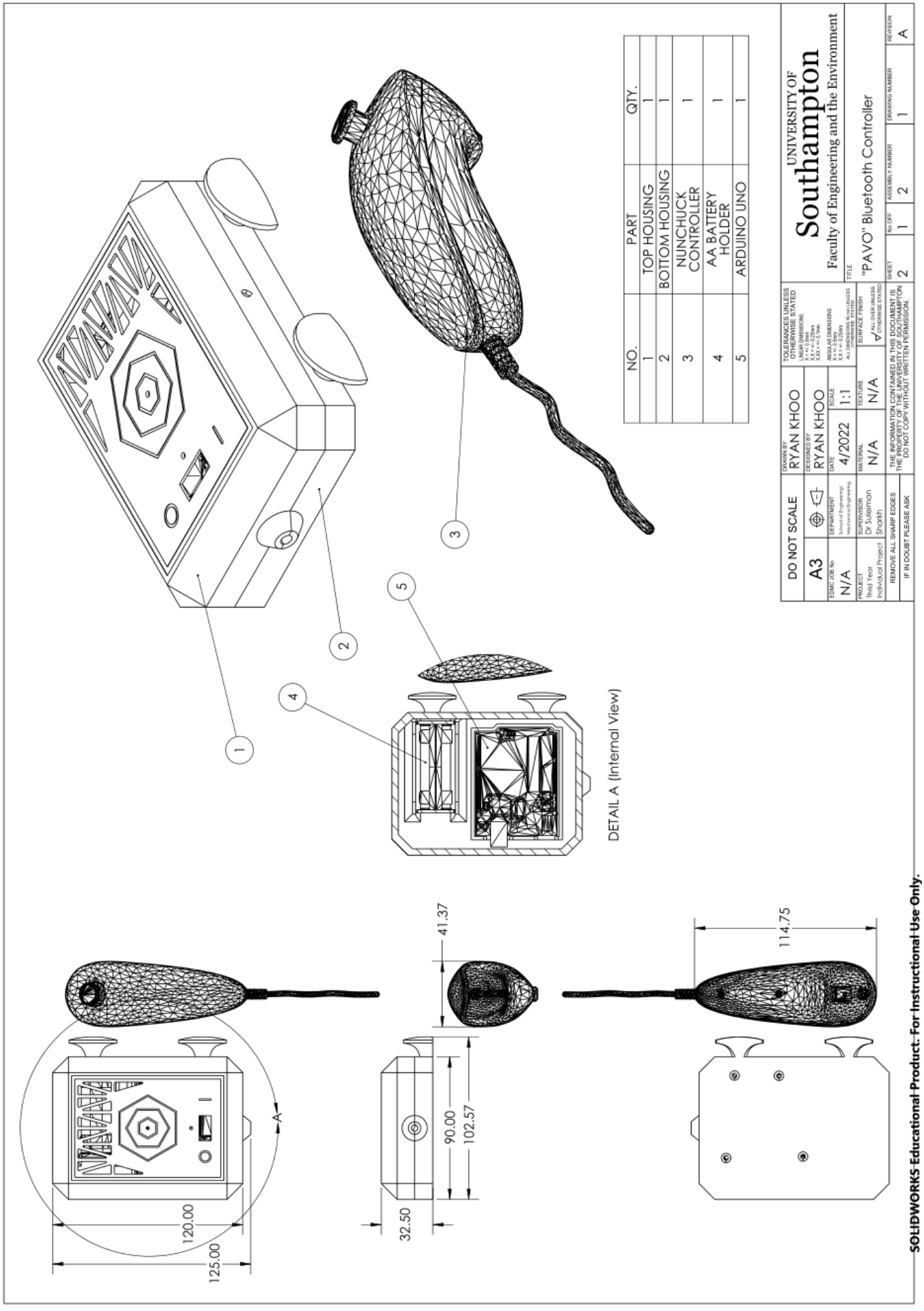
```
//delayed code loop, code below only occurs once every setDelay()
milliseconds to execute "edible" sets --------------------------------
```

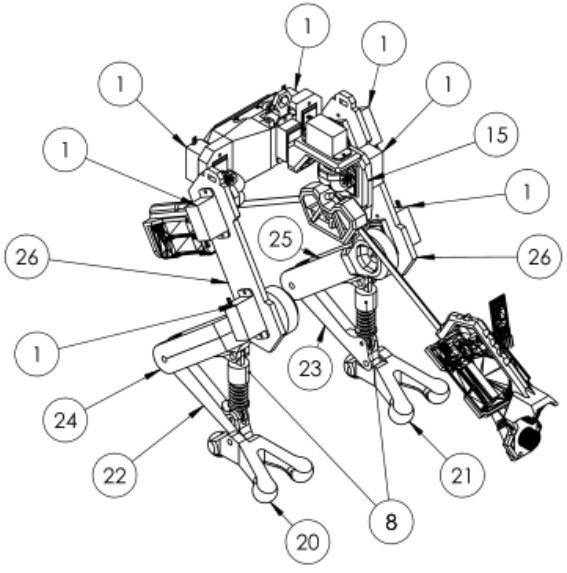
```
------------------------------------------------------------------------
---------
  //int currentTime = millis(); Already above
  //int timeOfLastLoop = 0; this line is in section 222222

  int timeSinceLastLoop = currentTime - timeOfLastLoop;

//Code that "eats" and executes from the pending set
  if (timeSinceLastLoop > setDelay -1){

    int availableSets = pendingSet[maxDivisions][8];// available set
determines both the number of sets inpending sets to be executed, and
the index of that set to be used (-1 because of zero indexing)
    if (availableSets != 0){
      executePosition(availableSets-1); //takes the most right-side
non-zero set and executes it
      //RYAN, you may need to delete the set that has been "eaten"
(15/3/22), nope, not needed (1/4/22)
      pendingSet[maxDivisions][8] = (pendingSet[maxDivisions][8]) - 1 ;
//number of "edible" sets now decreased by 1
//      Serial.print("delayed loop ran, and pending sets = ");
//      Serial.println(pendingSet[maxDivisions][8]);
//      Serial.print("executed position in pendingSets indexed at: ");
//      Serial.println((availableSets-1));

    } // end of if statement that eats pending set

    timeOfLastLoop = currentTime;
  }//end of delayed loop ----------------------------------------------
------------------------------------------------------------------------
-------------------------------------------------


//Manual Pitch Control-------------------------------------------------
----------------------

//RYAN add that this can only occur if pavo is in stationary mode and
break out of it back to normal balancing if it gets too unbalance while
playing with it
//manual pitch control
 if (state >= 110 && state <= 119){
  int pitchState = state - 110;
  int pitchDegree = map(pitchState, 9,0, (-
pitchRange+pitchCenter),(pitchRange+pitchCenter));
  pwm.setPWM(pitchServo, 0, anglePulse(pitchDegree));

 }
//Manual Roll Control
 if (state >= 120 && state <= 129  && cycleStage == 9){
  int rollState = state - 120;
  int rollDegree = map(rollState, 0,9, (-
rollRange+rollCenter),(rollRange+rollCenter));
  pwm.setPWM(rollServo, 0, anglePulse(rollDegree));
 }
 //Manual look Control
 if (state >= 130 && state <= 139  && cycleStage == 9){
```

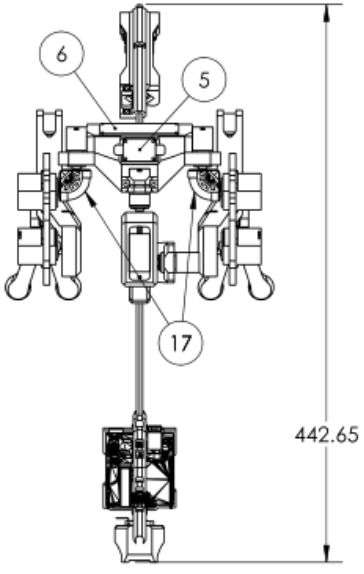
```
  int lookState = state - 130;
  int lookDegree = map(lookState, 0,9, (-
lookRange+lookCenter),(lookRange+lookCenter));
  pwm.setPWM(lookServo, 0, anglePulse(lookDegree));
 }
 //-------------------------------------------------------------------
----------------------

} // End of Void Loop


//                    ___________      .===.
//                                    /-=(o=`.
//                                   | ,__.-"``
//        __________                 _/ :  /
//                        _..--""`   :   /
//                     .-" : '     :    |
//    __________    ."   :        :     ;
//                .`   :        :       /
//          _,.-" :      .   :         /
//          `---'=-.,_;` '          ,='
//                         `:=-=.=-'====,_
//      _________          //`           `\\
//                        //
//                       //         run Pavo run
//                      `\=
```

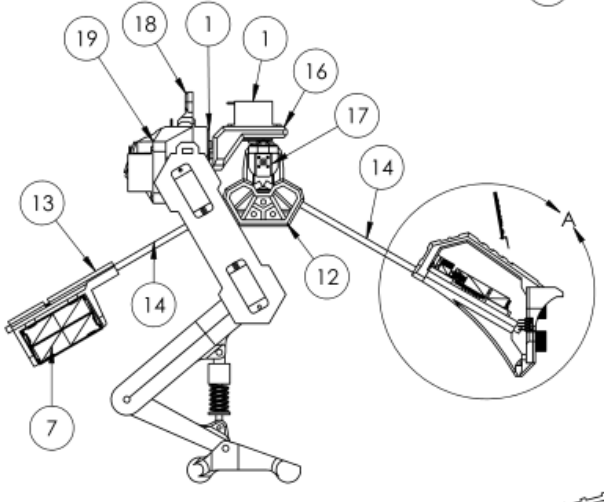
A.9 Engineering Assembly Drawings

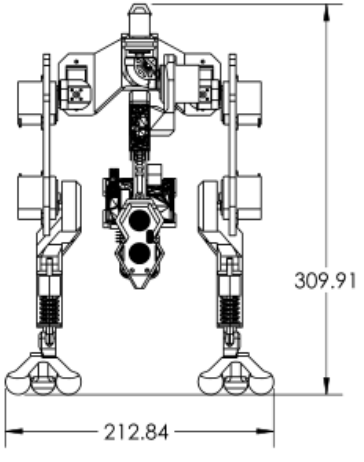
(Full resolution A3 drawings found on page 102 and 103, omit these for A4 printing)

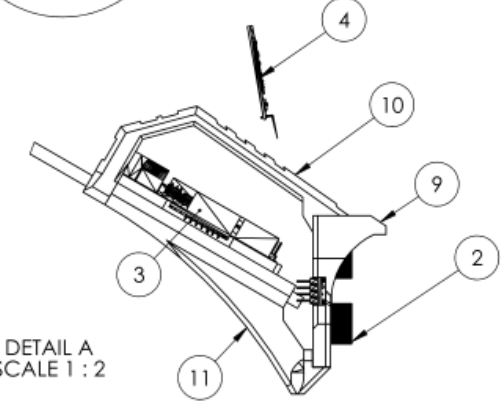










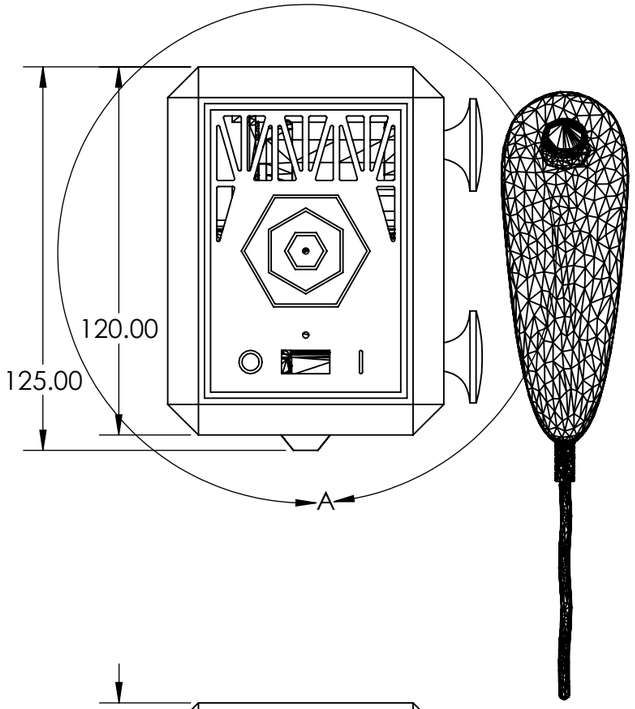


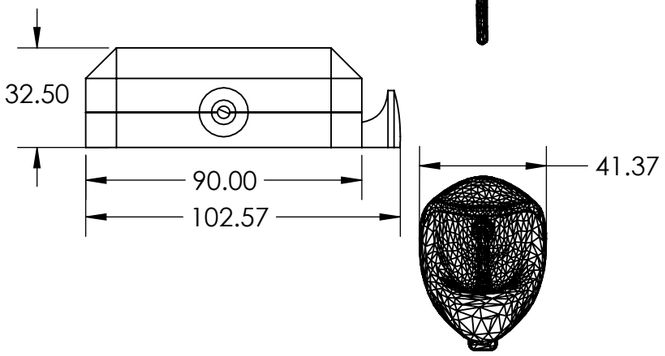
DETAIL A
SCALE 1 : 2

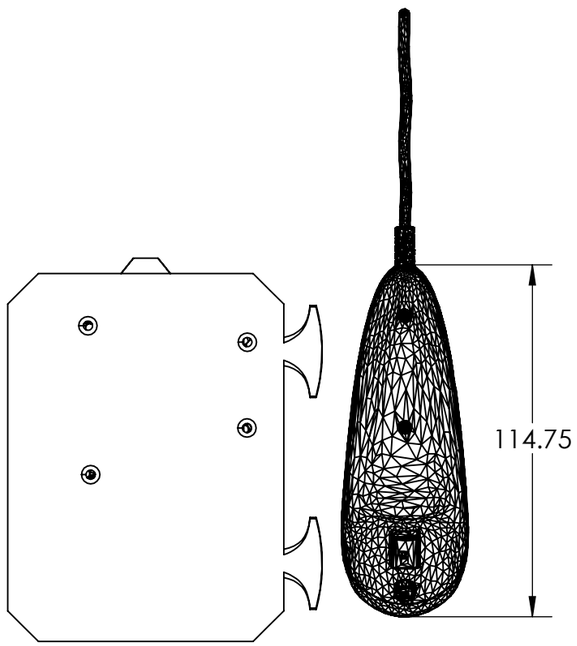
NO.	PART	QTY.
1	SER0056 Servos	9
2	HC-SR04 Ultrasonic Sensor	1
3	Arduino Uno	1
4	HC-05 Bluetooth Module	1
5	ICM-20948 9-DoF IMU	1
6	Adafruit 16-Channel 12-bit PWM Servo Driver	1
7	AA Battery Holder	1
8	75mm RC Damper Suspension	2
9	Face Plate	1
10	Head Comb Wire Rail/Stabiliser	1
11	Kneck Base/Arduino Mount	1
12	Spine Joint	1
13	Tail/Battery Mount	1
14	5mm Threaded Rod (23cm)	2
15	Spine Tilt/Pitch	1
16	Spine Swivel	1
17	90 Degree Servo Joint	3
18	Wire guide	1
19	Main Hip	1
20	Right Foot	1
21	Left Foot	1
22	Right Upper Foot	1
23	Left Upper Foot	1
24	Right Shin	1
25	Left Shin	1
26	Laser-cut Acrylic Thigh	2

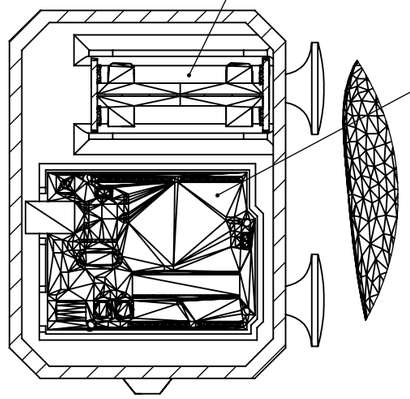
DO NOT SCALE		DRAWN BY: Ryan Khoo	TOLERANCES UNLESS OTHERWISE STATED LINEAR DIMENSIONS X = +/- 0.5mm, X.X = +/- 0.25mm, X.XX = +/- 0.1mm ANGULAR DIMENSIONS X = +/- 0.5mm, X.X = +/- 0.25mm ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED	UNIVERSITY OF Southampton Faculty of Engineering and the Environment	
A3		DESIGNED BY: Ryan Khoo			
EDMC JOB No: N/A	DEPARTMENT: School of Engineering/ Mechanical Engineering	DATE: 4/2022	SCALE: 1:4	TITLE: "PAVO" Bipedal Robot	
PROJECT: Third Year Individual Project	SUPERVISOR: Dr Suleiman Sharkh	MATERIAL: N/A	TEXTURE: N/A	SURFACE FINISH: ALL OVER UNLESS OTHERWISE STATED	
REMOVE ALL SHARP EDGES IF IN DOUBT PLEASE ASK	THE INFORMATION CONTAINED IN THIS DOCUMENT IS THE PROPERTY OF THE UNIVERSITY OF SOUTHAMPTON DO NOT COPY WITHOUT WRITTEN PERMISSION.	SHEET: 1	No OFF: 1	ASSEMBLY NUMBER: 1	DRAWING NUMBER: 1 / REVISION: B

Ryan Khoo Yeap Hong

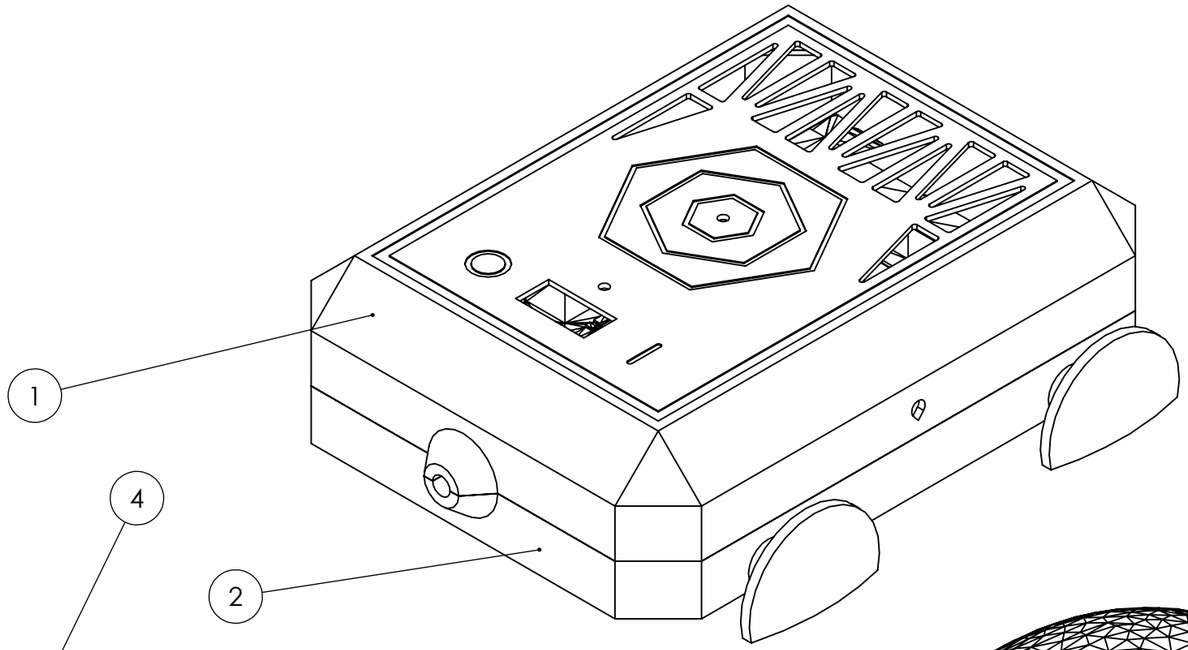


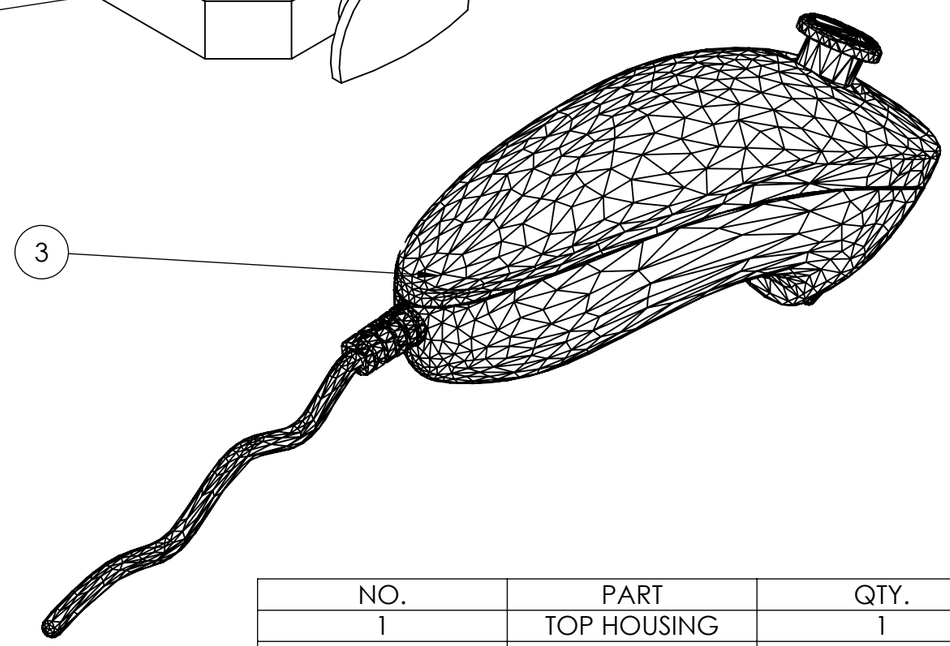






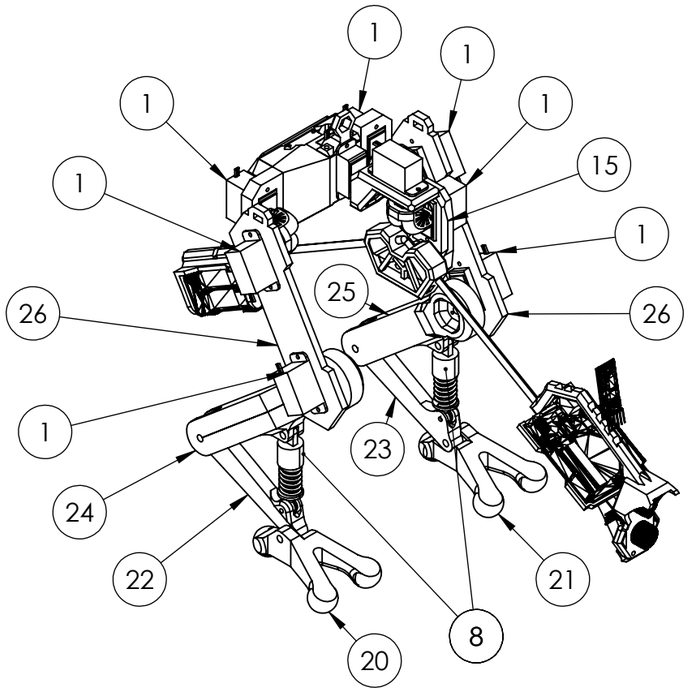
DETAIL A (Internal View)

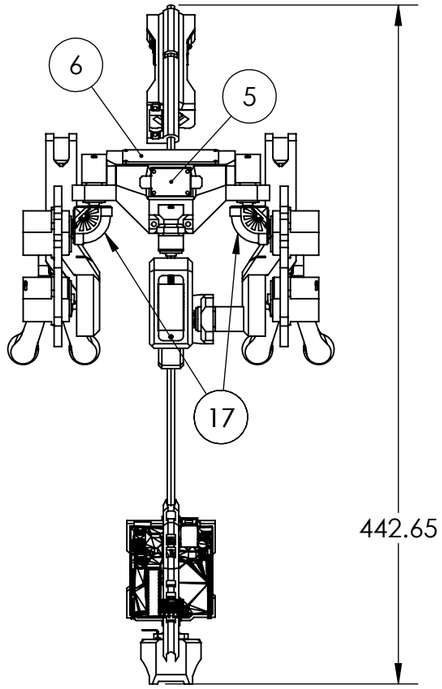


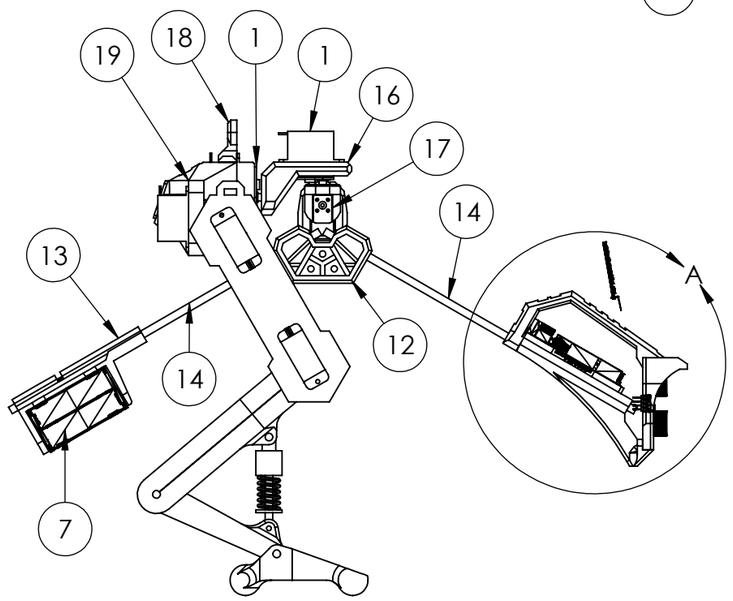


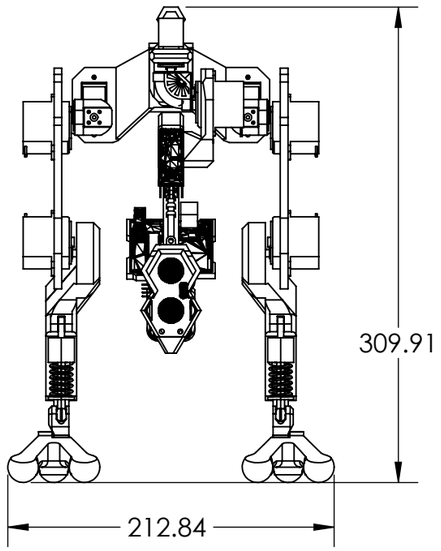
NO.	PART	QTY.
1	TOP HOUSING	1
2	BOTTOM HOUSING	1
3	NUNCHUCK CONTROLLER	1
4	AA BATTERY HOLDER	1
5	ARDUINO UNO	1

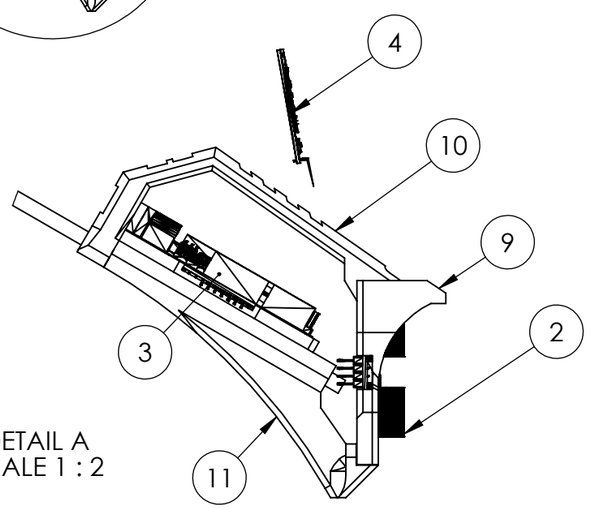
DO NOT SCALE		DRAWN BY: RYAN KHOO		TOLERANCES UNLESS OTHERWISE STATED: LINEAR DIMENSIONS X = +/- 0.5mm, X.X = +/- 0.25mm, X.XX = +/- 0.1mm; ANGULAR DIMENSIONS X = +/- 0.5mm, X.X = +/- 0.25mm; ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED	UNIVERSITY OF Southampton, Faculty of Engineering and the Environment
A3		DESIGNED BY: RYAN KHOO			TITLE: "PAVO" Bluetooth Controller
EDMC JOB No: N/A	DEPARTMENT: School of Engineering/ Mechanical Engineering	DATE: 4/2022	SCALE: 1:1		
PROJECT: Thrid Year Individual Project	SUPERVISOR: Dr Suleiman Sharkh	MATERIAL: N/A	TEXTURE: N/A	SURFACE FINISH: ALL OVER UNLESS OTHERWISE STATED	
REMOVE ALL SHARP EDGES; IF IN DOUBT PLEASE ASK	THE INFORMATION CONTAINED IN THIS DOCUMENT IS THE PROPERTY OF THE UNIVERSITY OF SOUTHAMPTON DO NOT COPY WITHOUT WRITTEN PERMISSION.	SHEET: 2	No OFF: 1	ASSEMBLY NUMBER: 2	DRAWING NUMBER: 1 / REVISION: A











DETAIL A
SCALE 1 : 2

NO.	PART	QTY.
1	SER0056 Servos	9
2	HC-SR04 Ultrasonic Sensor	1
3	Arduino Uno	1
4	HC-05 Bluetooth Module	1
5	ICM-20948 9-DoF IMU	1
6	Adafruit 16-Channel 12-bit PWM Servo Driver	1
7	AA Battery Holder	1
8	75mm RC Damper Suspension	2
9	Face Plate	1
10	Head Comb Wire Rail/Stabiliser	1
11	Kneck Base/Arduino Mount	1
12	Spine Joint	1
13	Tail/Battery Mount	1
14	5mm Threaded Rod (23cm)	2
15	Spine Tilt/Pitch	1
16	Spine Swivel	1
17	90 Degree Servo Joint	3
18	Wire guide	1
19	Main Hip	1
20	Right Foot	1
21	Left Foot	1
22	Right Upper Foot	1
23	Left Upper Foot	1
24	Right Shin	1
25	Left Shin	1
26	Laser-cut Acrylic Thigh	2

DO NOT SCALE		DRAWN BY Ryan Khoo		TOLERANCES UNLESS OTHERWISE STATED LINEAR DIMENSIONS X = +/- 0.5mm X.X = +/- 0.25mm X.XX = +/- 0.1mm ANGULAR DIMENSIONS X = +/- 0.5mm X.X = +/- 0.25mm ALL DIMENSIONS IN mm UNLESS OTHERWISE STATED	UNIVERSITY OF Southampton Faculty of Engineering and the Environment
A3		DESIGNED BY Ryan Khoo			TITLE: "PAVO" Bipedal Robot
EDMC JOB No: N/A	DEPARTMENT: School of Engineering/ Mechanical Engineering	DATE: 4/2022	SCALE: 1:4		
PROJECT: Thrid Year Individual Project	SUPERVISOR: Dr Suleiman Sharkh	MATERIAL: N/A	TEXTURE: N/A	SURFACE FINISH: ALL OVER UNLESS OTHERWISE STATED	
REMOVE ALL SHARP EDGES IF IN DOUBT PLEASE ASK		THE INFORMATION CONTAINED IN THIS DOCUMENT IS THE PROPERTY OF THE UNIVERSITY OF SOUTHAMPTON DO NOT COPY WITHOUT WRITTEN PERMISSION.			SHEET: 1 | No OFF: 1 | ASSEMBLY NUMBER: 1 | DRAWING NUMBER: 1 | REVISION: B