Mechanism design, kinematic analysis, simulation and implementation of a legged robot

Ethan Batt

School of Civil &. Mechanical Engineering Project Thesis
Curtin University

28th of October 2025

7A Forward Street,
Manning,
WA 6152

28th of October 2025

The Head
School of Civil & Mechanical Engineering,
Curtin University,
Kent Street,
Bentley,
WA 6102

Dear Sir,

I submit this thesis entitled “Mechanism design, kinematic analysis, simulation and implementation of a legged robot”, based on MXEN4000 Mechatronic Engineering Research Project 1 and MXEN4004 Mechatronic Engineering Research Project 2, undertaken by me as part-requirement for the degree of B.Eng. in Mechatronic Engineering.

Yours faithfully,

Ethan Batt
20563444

Acknowledgments

I would like to acknowledge my thesis supervisor Dr. Masood Khan for his support, guidance and patience throughout the completion of this project. Thank you for taking the time to point me in the right direction and provide me with valuable knowledge and suggestions for my thesis project.

I would like to thank all the teaching staff and students within the Mechatronics and Computer Science departments. Thank you for your friendliness and your assistance throughout my degree.

Finally, I wish to express my gratitude to my parents, partner and my family. I would not be where I am today without their unwavering love and support.

Abstract

This thesis project outlines the design, analysis and implementation of a modular quadruped robot developed for ease of maintenance and reconfigurability. This project follows a structured design process to achieve a final prototype of the robot that has been based heavily on past literature. During the design stage of this project the following processes were completed: the selection of an appropriate leg mechanism; definition of a gait sequence and foot trajectory; creation of a modular mechanical framework; completion of static, kinematic and dynamic analyses; development of an electronic subsystem; and integration with an open-loop control algorithm.

The resulting quadrupedal design is manufactured and assembled to verify part geometries. Once assembled the electronics and control method was systematically tested, before a dynamic motion test on various uneven surfaces was performed. Next, the theoretical carrying capacity of the robot was confirmed, and the results of its power consumption over a number of gait cycles for a variety of directions was obtained. The results of these tests show the project meets its defined scope and provides a working platform that can be extended in the future.

The following pages of this thesis were previously submitted as part of the Progress Report assessment for the MXEN4000 Mechatronic Engineering Research Project 1 unit: 1-2, 6, 11-18, 23-24, 26, 31-44, 56, 63-70, 102-117.

Nomenclature

Symbol	Meaning	Unit
x_m	Vertical position of the quadruped's foot	mm
y_m	Horizontal position of the quadruped's foot	mm
t	Time	s
h	Step height of the gait trajectory	mm
y_s	Perpendicular distance between the body and ground in the standing position	mm
θ	Horizontal phase shift	rad
β	Angular frequency	rad
x_s	Lifting point of the gait trajectory	mm
x_f	Landing point of the gait trajectory	mm
T	Duration of the gait trajectory	s
λ	Duty cycle of the swing phase	N/A
x_c	Horizontal position of the centre of the elliptical gait trajectory	mm
y_c	Vertical position of the centre of the elliptical gait trajectory	mm
ρ_1	Semi-major axis of the elliptical gait trajectory	mm
ρ_2	Semi-minor axis of the elliptical gait trajectory	mm
$\gamma(t)$	Orbital angle function of the elliptical gait trajectory	rad
l_1	Length of the fixed ground link of the five-bar mechanism	mm
l_2	Length of the second link in a clockwise direction from the fixed link of the five-bar mechanism	mm
l_3	Length of the third link in a clockwise direction from the fixed link of the five-bar mechanism	mm
l_4	Length of the fourth link in a clockwise direction from the fixed link of the five-bar mechanism	mm
l_5	Length of the fifth link in a clockwise direction from the fixed link of the five-bar mechanism	mm
A_2	Coefficient for the inverse kinematics equation for the second link	mm^2
B_2	Coefficient for the inverse kinematics equation for the second link	mm^2
C_2	Coefficient for the inverse kinematics equation for the second link	mm^2
A_5	Coefficient for the inverse kinematics equation for the fifth link	mm^2
B_4	Coefficient for the inverse kinematics equation for the fifth link	mm^2
C_3	Coefficient for the inverse kinematics equation for the fifth link	mm^2
φ_2	Angle of the second link anticlockwise from the positive x axis	rad
φ_3	Angle of the third link anticlockwise from the positive x axis	rad
φ_4	Angle of the fourth link anticlockwise from the positive x axis	rad
φ_5	Angle of the fifth link anticlockwise from the positive x axis	rad
ω_2	Angular velocity of the second link	$rad\ s^{-1}$
ω_3	Angular velocity of the third link	$rad\ s^{-1}$

ω_4	Angular velocity of the fourth link	rad s^{-1}
ω_5	Angular velocity of the fifth link	rad s^{-1}
α_2	Angular acceleration of the second link	rad s^{-2}
α_3	Angular acceleration of the third link	rad s^{-2}
α_4	Angular acceleration of the fourth link	rad s^{-2}
α_5	Angular acceleration of the fifth link	rad s^{-2}
φ_s	Angle of the line between the two passive joints connecting l_2 to l_3 and l_4 to l_5	rad
l_s	Length of the line between the two passive joints connecting l_2 to l_3 and l_4 to l_5	mm
F_R	Contact reaction force acting on the quadruped's foot	N
F_3	Resolved reaction force acting upon the third link	N
F_4	Resolved reaction force acting upon the fourth link	N
T_2	Articulated static torque acting upon the second link	Nm
T_5	Articulated static torque acting upon the fifth link	Nm
$\mu_{PID}(t)$	Output of a PID controller	N/A
$e(t)$	Error signal input of a PID controller	N/A
k_p	Proportional gain of a PID controller	N/A
k_i	Integral gain of a PID controller	N/A
k_d	Derivative gain of a PID controller	N/A
$M1$	Bottom left point of leg mechanism workspace	N/A
$M2$	Bottom right point of leg mechanism workspace	N/A
$M3$	Top right point of leg mechanism workspace	N/A
$M4$	Top left point of leg mechanism workspace	N/A
y_{MX}	Y coordinate of matching point of leg mechanism workspace	N/A
x_{MX}	X coordinate of matching point of leg mechanism workspace	N/A
d	Index of discrete interval of a gait cycle	N/A
$\sigma(d)$	Percentage of the resolution of the swing phase for interval	N/A
$\phi(d)$	Percentage of the resolution of the support phase for interval	N/A
F_f	Force due to friction	N
μ_s	Static coefficient of friction	N/A
J	Jacobian Matrix of the leg mechanism	N/A
J_q	Jacobian vector of joint angles	N/A
J_x	Jacobian vector of cartesian velocities	N/A
$F_2(X,Q)$	Transmission function for φ_2	N/A
$F_5(X,Q)$	Transmission function for φ_5	N/A

Table of Contents

Chapter 1. Introduction

1.1 Background

Legged robots have gained traction in recent years due to their exceptional ability to traverse rugged terrain, outperforming conventional wheeled and crawler robots [1, 2]. Typically, these highly mobile robots are divided into three categories, biped robots, quadruped robots and multilegged robots [1]. Quadrupeds emerge as the main category of legged robots, with greater stability and carrying capacity than bipeds, and better dynamic performance than multilegged robots [1-3]. This balance between agility and stability has contributed to their widespread development in both commercial and research fields, with their applications expanding across military settings, agricultural inspection, disaster recovery, and industrial uses [2].

1.2 Scope of project

This thesis aims to design, implement and test a quadruped robot capable of traversing uneven terrain. In this context, uneven terrain refers to moderately irregular natural and urban surfaces, such as grass, pavement, or wooden floors.

The robot will act predominantly as a reconfigurable and modular platform onto which manipulators, loads, or other tools can be fixed. The scope of development encompasses the mechanical design and validation of the robot's leg mechanism, integration of modularity between system components, design of a simplistic electronic subsystem, and a basic control system implementation. Advanced autonomy, perception, tool integration, accessory development and long-term deployment will be considered out of scope for this project.

The project aims to address the challenges, limitations and future work proposed by past literature. Due to the timeframe of this project the end solution will look to solve challenges and limitations brought through mechanical wear and tear, which impacts the robot's performance and operational longevity [2]. To do this it will incorporate a

modular approach to reduce maintenance complexity and increase universality as suggested in [4].

In order to quantify the success of the project, a number of project deliverables and a set of key performance indicators for the final solution have been developed.

Table 1: Project Deliverables

Identifier	Deliverable
D-1	**3D Models**
D-1A	3D model part for each 3D printed component.
D-1B	3D model assembly of a single leg module.
D-1C	3D model assemblies of integrated robot in at least two configurations.
D-2	**Analysis and Simulation**
D-2A	Static analysis of leg mechanism.
D-2B	Kinematic analysis of leg mechanism.
D-2C	Dynamic analysis of integrated robot through simulation.
D-3	**Electronic Subsystem**
D-3A	Schematic of implemented electronic subsystem.
D-3	**Control System**
D-3A	Source code for the implemented control system.
D-4	**Prototype**
D-4A	Physical prototype of designed robotic platform.

Table 2: Project Performance Indicators

Identifier	Key Performance Indicators
K-1	**Stability**
K-1A	The designed robotic platform must be able to stand freely without aids.
K-1B	The designed robotic platform must complete five gait sequences without falling.
D-2	**Movement**
K-2A	The designed robotic platform must traverse 1m in a forward and reverse direction.
K-2B	The designed robotic platform must complete a full left and right turn.
K-3B	The designed robotic platform must be able to carry 50% of its weight.
K-3	**Modularity**
K-3A	The designed robotic platform must be reconfigurable in at least two configurations.
K-4	**Electronics**
K-4A	The designed robotic platform must operate for a minimum of 30 minutes.
K-5	**Control System**
K-5A	The robotic platform must be controlled via a Bluetooth controller.

1.3 Project Outline

To commence this project a detailed literature review was conducted to evaluate existing quadrupedal robots, their design processes, and the technologies they use against the scope of the project. This review outlines the historical progression of legged robots, highlighting key advancements in technologies that have contributed to today's landscape. With this progression, the limitations of technology are explored, highlighting the issues due to wear and tear, maintenance and energy consumption, as well as challenges in the form of societal considerations.

Following an overview of the current landscape surrounding the development of these robots, a deep dive into their key components was completed. The evaluation of leg mechanisms forms a critical part of determining the functionality of these robots [5]. As such, leg mechanisms including prismatic, parallel, serial and compliant structures were investigated for suitability against this project.

The adjustment of these mechanisms forms another key part in the design of a quadruped, with the need to consider both the individual leg trajectories and the combined gait sequences. For this project simple trajectories such as sinusoidal and elliptical based paths were compared, whilst sequences for stable and slow movements were identified.

The incorporation of modularity into robotic systems was also investigated, with the effects of structures and the properties of mechanisms outlined in this review. Latticed based structures, with mechanical or magnetic properties were identified as prime candidates for this project.

The analysis of a quadruped's design is also a key subsystem of any robotic project. Static analyses can be conducted to verify the geometries and materials of these designs and can identify how static forces interact with the robot. Kinematic analyses on the other hand help determine the relationship between drive angles, positions, velocities and accelerations, whilst excluding the effects of dynamic forces [1]. Inverse and forward kinematics are often the result of such analyses and provide valuable uses for the control of the robot.

Existing implementations of electronic subsystems that form a part of quadrupedal robots were also evaluated. Within this evaluation the comparison of hobby grade

microcontrollers, including ESP32s and RasberryPis, was conducted. Different types of actuators and their drive mechanisms were also investigated, along with the use of sensors such as Inertial Measurement Units (IMUs) and cameras. The power sources and regulators also formed crucial parts of this review.

The final portion of the literature review outlines control systems to enable stable and versatile movement of quadrupeds. Within this section various open-loop and closed-loop algorithms are detailed.

Following the conduction of thorough research, the theoretical design and analysis of the quadruped robot developed in this project was conducted. This process began with the selection and derivation of parameters of a parallel leg mechanism. To achieve this, a dexterous workspace was defined based on existing implementations in literature.

This workspace could then be applied to generate a foot trajectory. From the evaluation of research, a sinusoidal gait was chosen and divided into discrete intervals using appropriate formulae. The individual paths of each leg could then be combined and offset in cyclical phase to achieve a walking gait sequence. Within this sequence each leg is lifted for only a quarter of its total duration, enabling the robot to maintain three points of contact at all times.

Next a modular system for the mechanical structure of the robot was developed. This system divided modules into component modularity, which standardises parts between modules, and operational modularity involving the separation of individual operational subsystems. Once a modular structure was defined, the 3D model of the entire robot could be developed and configured into two different arrangements.

With the completion of the mechanical design, a static, kinematic and dynamic analysis could be conducted on the robot. The static analysis allowed for the validation of Polylactic Acid (PLA) as the primary material for this project. The kinematic analysis identified the inverse and forward kinematic equations, applying them to perform a position, velocity and acceleration analysis of a single leg over its predefined workspace. The dynamic analysis identified the torque and force characteristics of each mechanism to assist with the selection of actuators.

After the verification of the mechanical structure, and the identification of loads required by the actuators, the electronic components could be selected. These components were predominantly chosen from their use in literature, with an ESP32

microcontroller, lithium-ion battery, servo motors and an IMU making up the key components. Once the components were selected, a schematic diagram could be created outlining the interaction between components.

The final portion of the design process was to implement an open-loop control system. This system leveraged third-party packages such as Bluepad32 to connect to peripherals and interact with the onboard electronics. The chosen control system works by phase shifting the individual leg trajectories to form a walking gait, with any adjustments to this gait made by the user via the remote.

To test the outlined quadruped robot, a prototype was manufactured and assembled, verifying the geometries of the design. This prototype was then subject to a variety of tests for comparison against the project performance indicators. These tests included power and control system verification, dynamic motion evaluation, carrying capacity validation and energy consumption performance.

The results of the testing indicated that the designed robot closely met the scope of the project. The discussion of future work to extend the project outlines the addition of cameras and the inclusion of closed-loop control.

The design methodology and steps of this project is broadly applicable to the development of almost all robotic and mechatronic systems. The systematic framework of conducting detailed research, developing and analysing a theoretical design, and implementing and testing a prototype, will more times than not, lead to the development of a successful system.

Chapter 2. Literature Review

2.1 History of Legged Robots:

One of the first examples of a quadruped robot is the Walking Truck, developed by the General Electric Company in the 1960s [2, 3, 6]. It employed a manually driven, hydraulic system to achieve basic locomotion over uneven terrain, whilst carrying loads up to 200kg [3]. Although this approach was highly inefficient, it marked a significant milestone in the development of quadruped robots, demonstrating that this form of four-legged locomotion was feasible [3].

The next key milestones in the development of quadruped robots came in the 1970s, with the release of KUMO-I and PV-II. KUMO-I marked the first use of spider-like legs in a quadruped robot, signifying a significant leap in leg mechanisms for these robots [6]. PV-II, the successor of KUMO-I, became the premier robot to navigate stairs via sensor-based motion [6]. In the 1980s, Titan III built upon this climbing success, becoming the first climbing robot with intelligent programming [6]. To achieve this Titan III utilised a pantograph leg mechanism, an attitude sensor and toe tactile sensors on each leg [6].

Arguably the largest development in the legged robotics space came in 2005, with the introduction of Boston Dynamic's Big Dog. This quadruped reached unprecedented stability and mobility, reliably handling uneven terrain and implementing autonomous balancing for disturbance recovery [2]. To achieve this feat, BigDog utilised advanced onboard computing, sensors and hydraulic actuators, with the goal of carrying equipment in military environments. The main factor contributing to this large leap, is the significant advancements in microprocessor technology during the 2000s [2]. The success of BigDog led Boston Dynamics to release its successor Spot in 2015, which utilised electrical actuators [2]. As one of the first commercialised robots, Spot marked a shift away from the academic development of legged robots and towards commercialisation [2].

The last decade has seen the rapid advancement of quadruped robot platforms developed for generalised applications. In 2016, ETH Zurich released ANYmal, a quadruped designed for inspection tasks across the offshore, industrial and mining

sectors [2]. ANYmal focused heavily on advanced perception and automation to support its modular joint design, which could achieve full rotation, switching between kinematic configurations [2, 3]. MIT's Cheetah series is also a notable mention in the legged robotics field, focusing primarily on high-speed locomotion, their Cheetah 3 robot achieved a joint velocity of 21 rad/s, almost double that of ANYmal's [3, 7]. 2019 saw the release of HyQ2Max, a hydraulic based quadruped capable of pulling a 3.3 tonne airplane, revolutionising the loading limits of four-legged robots [3].

2.2 Challenges and Limitations

Throughout literature, challenges and limitations of past and present implementations of legged robots have been documented. These challenges and limitations can be divided into five overarching categories: terrain navigation, energy consumption, robustness, functionality and general societal concerns.

2.2.1 Terrain Navigation

One of the most significant of these challenges is the reliable navigation of highly variable and extreme environments [2]. Surface conditions such as sand, gravel, mud and ice remain a challenge in terms of traction and balance, often causing slippage and falls in legged robots [2]. These conditions expose ongoing limitations in robot movement algorithms, requiring real time adjustments of gaits through accurate perception of the surrounding environment.

Additionally, large objects, including rocks, logs, gaps and steep inclines add to the challenge of terrain navigation [2]. The physical limitations of quadrupedal robots, including restricted joint articulation and fixed limb lengths, constrains the robot's ability to overcome these obstacles [2]. Although progress has been made through automated leg reconfigurations, the physical limitations still hinder their performance in most cases [8].

Environmental conditions further complicate these difficulties with terrain navigation. Variations in weather such as moisture, dust and extreme temperatures can interfere

with sensors, reducing the accuracy of terrain maps and subsequent foot placements [2, 9]. These conditions also result in the rapid degradation of battery life and motor efficiency, requiring robust hardware [2].

2.2.2 Energy Consumption

Energy consumption represents another key limiting factor of robotic systems that extends into legged robots. With the evolution of complex motion algorithms, advanced sensing capabilities, computational loads and high-performance actuators, the demand for power increases accordingly [2]. Combined with the constraints on available power sources and the limitations of battery technology, energy consumption becomes a significant challenge to quadruped robots [9].

The combined effect of high-power demands and restricted capacity results in shortened operational periods and frequent battery replacement [2]. As a consequence, the robot's ability of completing extended missions, particularly in extreme environments, has been effectively eliminated [2]. As a compromise to energy consumption, battery capacity may be increased to allow for greater operational periods, however, this generally results in additional robot weight, which negatively impacts, agility, mobility and energy efficiency [2].

Recent innovations, such as the use of elastic components to store mechanical potential energy, has taken steps forward to improve overall energy consumption [2]. Despite these advancements, energy consumption still remains a core challenge for legged robotics.

2.2.3 Robustness

The mechanical and electrical robustness of a quadruped robot remains a substantial challenge to its operational reliability. Components such as actuators and joints often experience significant impact forces and repetitive stress throughout locomotion [2, 9]. These external forces accelerate the degradation of components, often resulting in

premature failure that compromises the robot's operational longevity and performance [2].

To combat deterioration, frequent maintenance is typically performed, completing routine checks and replacing parts to improve overall reliability. However, these procedures introduce further complications. They are often time consuming, labour intensive and costly, particularly when robots have not been designed for ease of maintenance. These issues are compounded in remote and hazardous environments, where access to specialised components and tooling may not be readily available [2].

A further limitation in the robustness of quadruped robots is their inherit lack of ability to handle or recover from component failures, which often leads to complete system shutdowns [2]. To address vulnerabilities, robots should be developed to implement redundancy and fault recovery, improving overall robustness for hardware malfunctions [2].

2.2.4 Societal Considerations

As legged robot technologies become widely accepted in industry, the social and ethical implications of their deployment must be addressed with caution and care [9]. These concerns typically arise from the safety, privacy, and potential for job displacement of the wider community [9].

One major safety risk is the potential of human-machine collisions. These risks are exacerbated with the implementation of autonomy, where errors in human detection, unpredictable behaviour due to sensor failure and software bugs could lead to injury [2]. Given the high-power density of quadruped actuators, erratic malfunctions or collisions may result in severe physical harm [2]. Furthermore, the response of robots in emergency situations may enhance the safety risks associated with the technology [2]. Typically, quadrupedal robots require human interaction to complete critical decision making, particularly in events such as fires, which adds complexity to existing hazardous situations [2]. To ensure safe interactions with quadruped robots, algorithms should be set in place in the event of emergency situations.

An ethical concern surrounding quadruped robots is the threat of invasion of privacy that they may present [9]. Often, robots are equipped with cameras and sensors that

record large amounts of data regarding the surrounding environment [2]. If this data is mismanaged or improperly regulated, intellectual property or individual data may be collected without consent [2]. The advanced technology and remote nature of these robots exposes vulnerabilities to cyberattacks, under which sensitive information may be obtained [2]. As a result, regulations must be put in place to limit exposure of sensitive information and address privacy concerns.

Finally, the implementation of legged machines is likely to disrupt employment in industry, as automation replaces low-skilled labour tasks that have traditionally been performed by humans [2]. This creates a disproportionate effect on the employment landscape, with workers who accommodate lower skill roles losing job opportunities, whilst the demand of higher skilled labour increases [2]. To mitigate community disruption, retraining programs and educational initiatives should be implemented to stabilise the workforce [2].

2.2.5 Functional Limitations

The advancement of quadruped technology has come leaps and bounds in recent years, with robots such as ANYmal achieving success in specific functionalities such as inspection, and MIT's cheetah reaching highspeed locomotion [2, 3, 7].

With the success of specific functions, literature outlines a need for legged robots to become multifunctional and adaptable, in order to adapt to various environments and complete a number of tasks [10].

A promising strategy to address this limitation is the adoption of a modular design approach. Robots could implement a modular design allowing for the interchangeability of components such as leg mechanisms and body structures [4]. This modularity helps promote the robot's flexibility, expandability, reconfigurability, universality and reduce maintenance complexity [4].

2.3 Leg Mechanisms

The mechanical design of a quadruped robot's leg mechanism plays a vital role in determining the application of the robot, its performance, adaptability, locomotion speed and total load capacity [5]. Given the importance of this subsystem, research has been conducted to identify typical leg configurations, their advantages and their limitations.

2.3.1 Prismatic Leg Configurations

One of the simplest leg configurations utilised in a legged robot is the prismatic leg. This configuration consists of two degrees of freedom (DOF): a singular revolute joint at the hip, and a linear prismatic joint used to adjust the length of the leg [5]. A depiction of this leg configuration is given below in Figure 1.

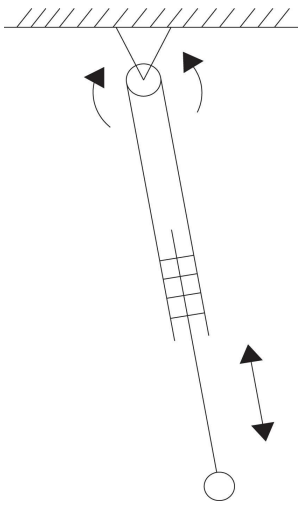

Figure 1: Prismatic Leg Configuration, Sourced From [5]

Prismatic legs are among the lightest and simplest mechanisms, displaying low inertia, which allows for a wide selection of actuators and the implementation of a basic controller [5]. These leg configurations also achieve high response speeds and efficient space utilisation, however, due to their fewer rotating joints, they fail to sufficiently adapt to uneven terrain [3, 5].

2.3.2 Serial Leg Configurations

Serial or articulated leg mechanisms are the most prominent configurations used in quadruped robots [5]. These mechanisms utilise multiple revolute joints to achieve a highly variable leg length and dexterous workspace, whilst striking a balance between control complexity, stability, performance and terrain adaptability [5, 11]. A depiction of a serial leg configuration is given in Figure 2 below.

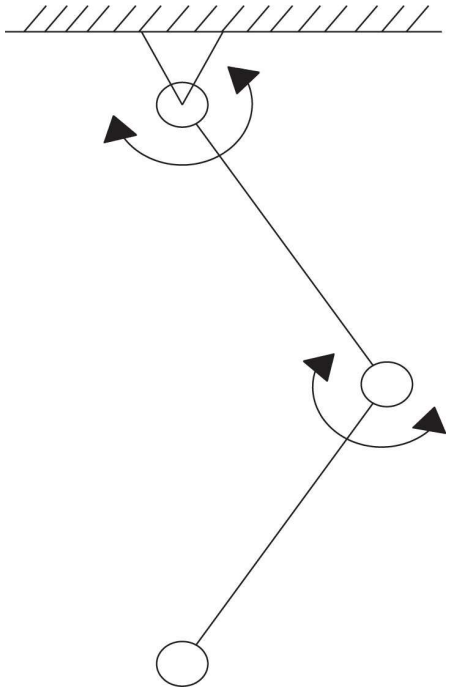

Figure 2: Serial Leg Configuration, Sourced From [5]

Serial mechanisms can be further broken down into three key categories: mammal-type articulated legs, sprawling-type articulated legs and redundant articulated legs [5].

Mammal-type articulated legs are positioned such that the first segment of the leg is directed vertically downwards when in the standing position. The leg consists of three rotational DOF: the hip abduction adduction (HAA) joint, the hip flexion extension (HFE) joint and the knee flexion extension (KFE) joint [1, 5]. The topology for a quadruped robot with mammal-type articulated legs is given by Figure 3 below.

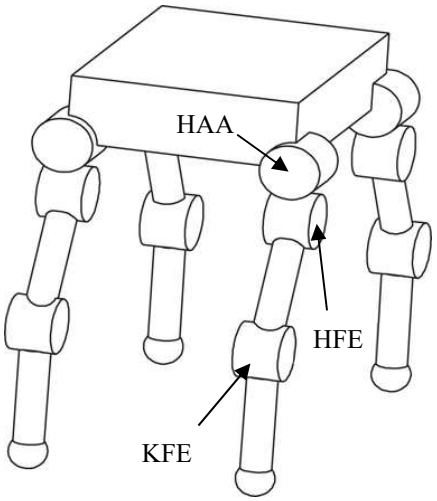

Figure 3: Mammal-Type Serial Leg Configuration, Sourced From [5]

Mammal-type leg mechanisms experience little joint torque when bent and maintain a small form factor allowing operation in narrow environments [5].

Serial leg configurations in which the first leg segment extends in a horizontal outwards position are known as sprawling-type articulated legs [5]. Much like mammal-type legs, this design consists of three rotational DOF: a hip yaw joint, a hip pitch joint and a knee pitch joint [5]. The topology for a quadruped robot with sprawling-type articulated legs is given by Figure 4 below.

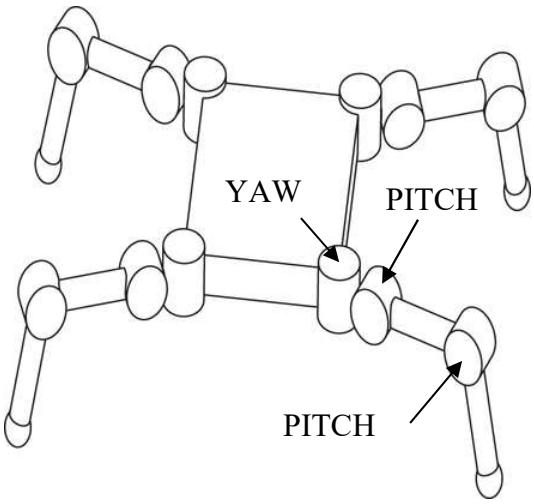

Figure 4: Sprawling-type Serial Leg Configuration, Sourced From [5]

The advantage of this type of leg structure is the ability to lower the robot's centre of gravity, allowing for high stability [5]. The configuration is limited, however, by a larger footprint, hindering its operations in a narrow space.

Redundant articulated mechanisms aim to mimic hoof or toe legs observed in nature [5]. These legs typically contain four rotational DOF: a HAA joint, a HFE joint, a KFE joint and an ankle flexion extension (AFE) joint. A depiction of this leg configuration is given below in Figure 5.

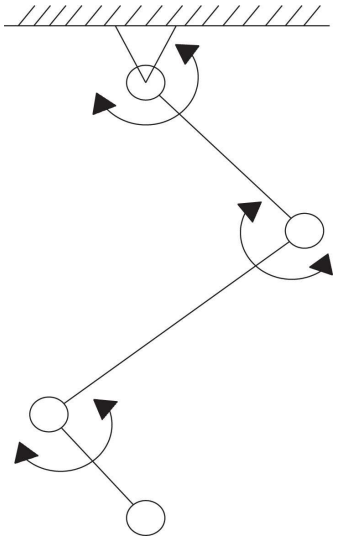

Figure 5: Redundant Serial Leg Configuration, Sourced From [5]

Due to their biomimicry, this configuration achieves better biometric and kinematic properties over complex terrain [5]. The extra degree of freedom in this configuration does, however, produce higher demands for control, sensing, structure and increases inertia.

2.3.3 Parallel Leg Configurations

Although serial leg mechanisms are common in industry and academic research, they present several challenges. One of these challenges is the accumulation of weight at each joint along the kinematic chain, as each actuator must support the weight of subsequent links and actuators [1]. This can limit the load capacity of the robot and increase the inertia of the mechanisms if actuators are placed at each joint [1]. To address this issue, transmission systems are often utilised to drive lower joints, however, this can reduce the precision of the mechanism [1]. As an alternative, parallel mechanisms implement a closed loop linkage, allowing for the actuators to be placed in the body of the quadruped without the need for transmission systems [1]. A depiction of one variant of a parallel leg configuration is given below in Figure 6.

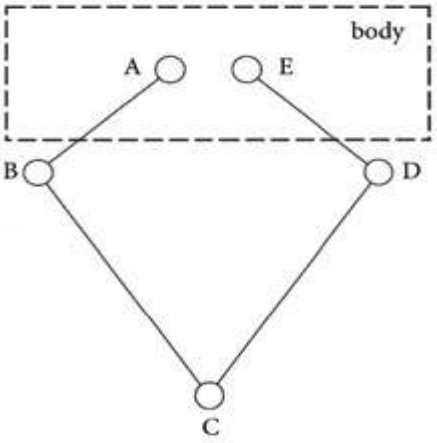

Figure 6: Parallel Leg Configuration, Sourced From [1]

This design substantially reduces the inertia of the links, providing higher stiffness, load capacity and greater energy efficiency [1, 11, 12]. Parallel mechanisms also eliminate a DOF by implementing differential drive instead of a HAA joint for steering, simplifying the control system [1]. A notable drawback of the configuration in comparison to its serial counterpart is a noticeably smaller reachable workspace [13].

2.3.4 Compliant Leg Configurations

Similar to prismatic legs, compliant leg mechanisms provide a simple configuration [2]. Rather than a linear prismatic joint, these mechanisms make use of passive elastic elements to store energy and reduce stress [2]. Although an improvement upon prismatic legs, compliant configurations are strictly limited to passive adaption to uneven terrain, instead of active joint movement [2]. A depiction of an array of compliant leg mechanisms is given below in Figure 7.

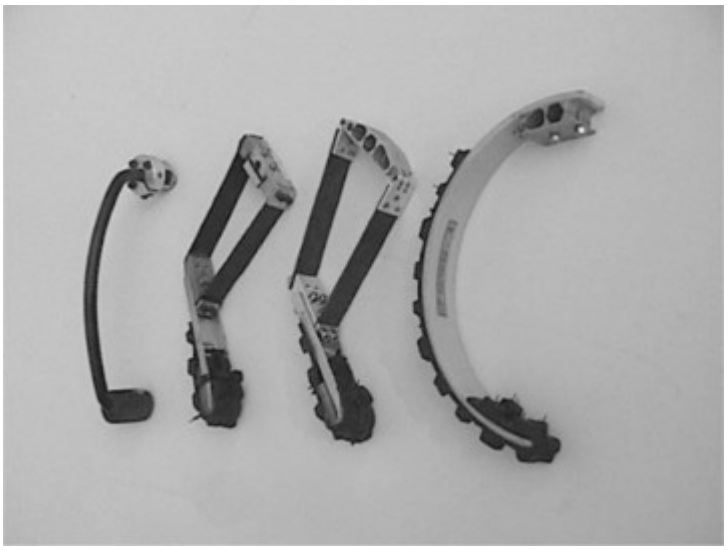

Figure 7: Compliant Leg Mechanisms Utilised by RHex, Sourced From [14].

2.4 Gait

Quadrupedal robots have the ability to adjust their leg mechanisms to a variety of joint configuration sets depending on environmental conditions and required actions [12]. These joint sets are often referred to as gait trajectories and can be combined to form a gait sequence [12]. Effectively controlling these trajectories and sequences enables the robots to achieve stability, terrain adaption, flexibility, and kinematic adjustment [12]. For these reasons, the gait of a robot is a key element in its design.

2.4.1 Gait Trajectory

As mentioned, the gait trajectory of a robot is a type of joint configuration set that defines its movement, often referring to the path taken by the foot of the quadruped. These trajectories range from simple geometric shapes along a two-dimensional plane to complex optimised paths that operate in three-dimensional space. For the implementation of this project, only the basic geometric paths are considered, specifically, sinusoidal, cycloid and elliptical trajectories. These form factors typically define the swing phase, a portion of time in which the foot is lifted from the ground [1]. The complementary portion of this trajectory is known as the support phase, which generally forms a horizontal line, during which the foot maintains contact with the ground. Through the consultation of literature, the formulae for each swing path can be determined. These are as follows.

The equation for a sinusoidal swing phase used by [15] and defined in [16], is given by:

$$y_m = h\sin(\beta x + \theta) + y_s \tag{1}$$

The equation for a cycloidal swing phase as defined in [1] is given below, where *t* is bound from zero to an end time *T*:

$$\begin{cases} x_m(t) = \left(x_f - x_s\right) \cdot \frac{\frac{2\pi t}{\lambda T} - \sin\left(\frac{2\pi t}{\lambda T}\right)}{2\pi} + x_s \\ \quad y_m(t) = h \cdot \frac{1-\cos\left(\frac{2\pi t}{\lambda T}\right)}{2} + y_s \end{cases} \tag{2}$$

The equation for an elliptical swing phase defined in [17] is given below, where *t* is bound from zero to an end time *T*:

$$\begin{cases} x_m(t) = \rho_1 \cdot \cos\left(\gamma(t)\right) + x_c \\ y_m(t) = \rho_2 \cdot \sin\left(\gamma(t)\right) + y_c \end{cases} \tag{3}$$

Each of the defined geometric paths offer distinct advantages which should be considered when designing a gait trajectory. Elliptical paths are often used *in situ* with cubic polynomials, enabling precise speed control over the path to achieve a low lifting and landing velocity [17]. Cycloidal paths offer similar benefits and are found to reduce impact forces when landing, giving greater stability in motion [1]. Sinusoidal trajectories are a proven gait trajectory for parallel leg mechanisms, with the path being utilised in Stanford Doggo and Minitaur to achieve stable gaits with an open loop controller [15].

2.4.2 Gait Sequence

Sequence planning or gait pattern, describes the movement timings between limbs, highlighting the transitions of each leg between its support and swing phase [1]. These sequences are often depicted as fractions or percentages of a gait cycle, and significantly influence the movement speed and stability of a quadruped [12]. Exemplar gait patterns are shown in Figure 8, with *LF* referring to the front left limb, *RH* referring to the back left limb, and so on.

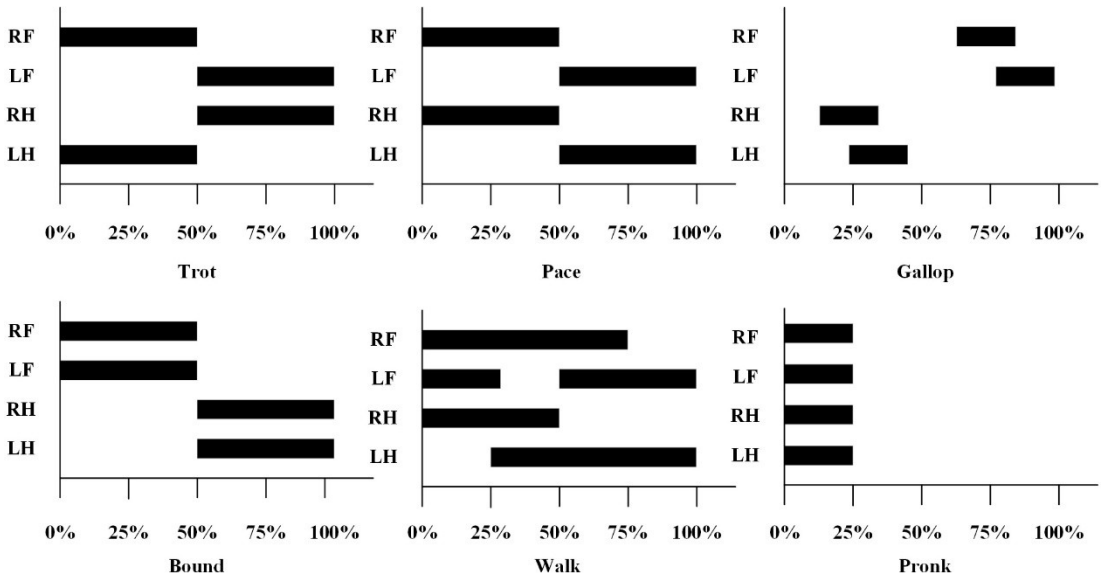

Figure 8: Common Gait Patterns, Sourced From [18].

2.5 Modularity

Modularity in this project forms the core of its innovation, as such, a literature review was conducted to determine a suitable implementation. This review is broken into three subcomponents: the overarching arrangement of modules in robotic systems, the connection types employed between modules and the connection mechanisms used to join them.

2.5.1 Module Structure

The topological architecture of a modular system defines the structure formed by the connection and organisation of individual modules in space. The types of structures formed by traditional modular robots can be broken down into three key sections: chain type, lattice type and hybrid type structures [19-21].

Chain type architectures describe the connection between individual modules to form single or multi-branched linkages known as linear or tree type topologies [19, 20]. These linkages form a serial architecture that can be positioned freely in three-dimensional space, allowing them to extend to bridge length, or fold to occupy space [19, 20]. Generally, modular robots that implement chain-type architectures result in a snake or caterpillar-like configuration, allowing them to form wheels to cover distance more rapidly, or, thin elongations to pass through tunnels [19, 20]. Notable past implementations of this topology includes the CONRO system (depicted in Figure 9 below), PolyBot and Molecule [19-21].

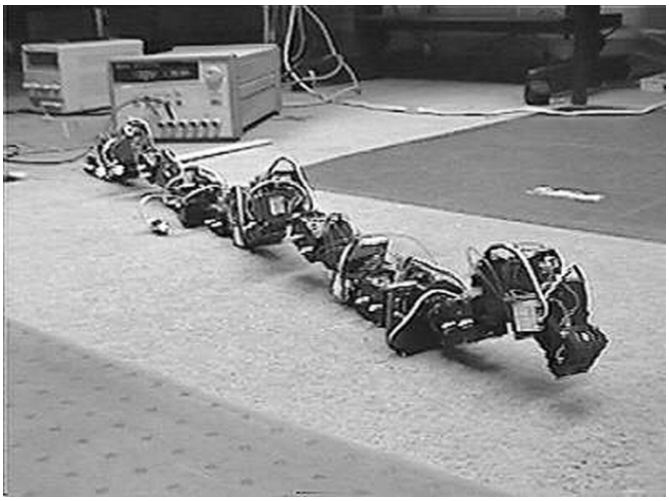

Figure 9: CONRO Project Chain-like Module Architecture, Sourced From [22].

Unlike chain systems, lattice type structures are distinguished by constraining modules such that they occupy discrete locations, forming regular three-dimensional patterns including cubic or hexagonal grids [19, 20]. This form of architecture results in densely packed shapes that are indicative of crystalline structures [21]. A subset of lattice structures are truss-based systems that utilise scaffolding in the form of struts or linkages to connect modules [19, 21]. Examples of successful implementations of lattice type modular systems are I-Cube, Crystalline and Robot Pebbles (depicted in Figure 10 below) [19-21].

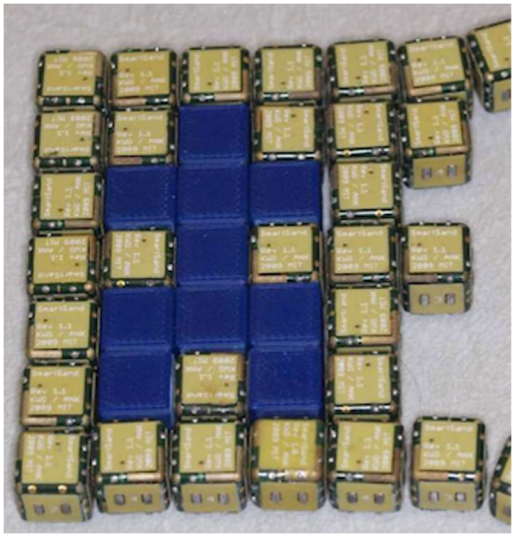

Figure 10: Robot Pebbles Lattice-like Module Architecture, sourced from [23].

The final topological structure for modular systems is hybrid-type architectures that combine both chain and lattice structures [20]. Modules utilising this category of architecture are able to arrange themselves in snake-like chains or block-like lattices [20]. This form of structure provides, greater levels of locomotion at the cost of more complex nodes [21]. M-Tran, SMORES and UBot are prime examples of hybrid modular systems [19-21].

2.5.2 Module Connection Types

The coupling of modular robotic systems is critical to the overall functionality of the system [24]. Each coupling must ensure strong, rigid connections whilst balancing size, complexity and in some cases, power consumption [20, 21, 24]. The coupling mechanisms for physically joining individual modules can be classified by the forces they employ to form the connection between interfaces [24]. These classifications are mechanical, magnetic, adhesive and vacuum connectors [25].

Mechanical connection types rely on forms of mechanical locking to achieve robust joining between module interfaces [24]. They can often be divided into two physical categories: rigid and soft connections [25]. Rigid connectors allow for high connection strength and precise alignment by implementing rigid features such as snap lock mechanisms [25]. Due to their rigid mating surfaces, these connectors also enable the transfer of power and data through the connector, allowing the formation of ‘plug and play’ modules [25]. Soft connectors on the other hand, make use of compliance in flexible materials to join modules [25]. These couplings allow for flexure about the coupling interface, however, are limited by low connection strengths [25].

A coupling that makes use of permanent or electromagnets belongs to the category of magnetic type connectors [20, 24]. A primary benefit of these forms of connections is their support for miniaturisation, enabling modularity on even micro-scale systems [24]. Furthermore, magnetic connectors are inherently self-aligning which helps reduce the complexity of joining procedures [24]. A downside of these couplings is the need for actuation when utilising electromagnets which can increase the power requirements of the system [25].

Adhesive type couplers bond modules by utilising materials such as glue, viscoelastic and electrostatics [25]. Although these methods offer high connection strength, they are often one-off and irreversible which hinders reconfigurability [25].

Vacuums are the final form of connection mechanisms that utilise suction forces to join modular systems [25]. This mechanism allows for rapid connection and disconnection with reasonable connection strength, however, requires precise manual alignment and can drive up power consumption [25].

2.5.3 Connection Mechanisms

Following the classification of couplers based on the primary employed joining force, they can be further categorised into individual connection mechanisms.

Pin-hole configurations make up the first of these categories and are one of the most established mechanical couplers [24]. These mechanisms rely on a set of holes and corresponding pins on opposite connection interfaces to enable initial meshing of the connectors [24]. With the simplest form of this coupler relying on the friction between connectors, these mechanisms typically necessitate additional locking components to prevent disengagement [24]. One of the most popular locking components is Shape Memory Alloy (SMA) wire utilised by the CONRO and PolyBot systems [24].

A similar connection mechanism is the hook coupler which utilises hooks or grippers with corresponding holes or grooves to join modules [24]. These designs are greater in complexity than pin-hole configurations, however, they omit the need for a locking mechanism, facilitating connection in a singular motion [24]. M-TRAN III and Roombot are primary examples of robots that utilise this form of connection [24].

Lock and key mechanisms are characterised by their use of unique shapes that are inserted into a receptacle hole and rotated to prevent disengagement [24]. Much like hook mechanisms, this coupler facilitates connection in a single motion [24]. It is critical these types of connectors allow for tolerance around the unique keys and cavities that enable rotation, to account for misalignment that could otherwise complicate reconfiguration [24]. Noteworthy examples that make use of this mechanism are Crystaline and I-Cube modular systems.

2.6 Static Analysis

In literature the mechanical analysis for a legged robotic system typically begins with a static analysis of a single leg mechanism. This analysis aims to determine the external forces and moments acting upon the body, as well as the internal stresses and strains experienced by the material. In [26], a static analysis was conducted using finite element analysis (FEA) to verify the mechanical design and material selection of the developed robot. In this study, polylactic acid (PLA) was employed as the material of the mechanism, and an external load equivalent to five body weights was applied [26]. The results of the study were used to identify the key areas of stress experienced in each leg component and verified the material's ability to withstand the applied load.

A different approach was taken in [27], which analysed a hexapod robot during its support phase whilst traversing a slope. This approach analyses a static equivalent model to determine the static articulated torques at each joint in terms of the joint angle [27]. To do this, the self-weight and external load are resolved to a point force acting at the robot's centre of mass [27]. The moments and forces created from the resulting contact force is then balanced to determine the relationship between joint angles and these dynamics. The final output is plotted as a torque, or contact force surface, which informs the reduction of the robot's mass and energy dissipation, as well as the selection of viable actuators [27].

2.7 Kinematic Analysis

During kinematic analysis, the position, velocity and acceleration changes in a mechanism are identified, without considering the applied force [1]. Kinematic analyses generally consist of two components: inverse kinematic equations and forward kinematic equations, which enable the calculation of the foot position and joint angles of the robot respectively [2]. These equations are critical to maintain balance, follow precise trajectories and achieve effective motion planning [2].

The kinematic analysis often requires the derivation of equations using the leg mechanism's geometries. For well-established mechanisms such as the planar five-bar

linkage, this derivation has been completed by previous literature. This is the case for the inverse kinematic equations of the leg mechanism for this project, whose generalised form are derived in [28].

Building upon this literature, the geometric parameters can be applied to obtain the inverse kinematic equations for this project, enabling the identification of the system's joint angles from a cartesian foot position. To obtain the position, velocity and acceleration relationships of the mechanism, forward kinematic analysis can be applied.

This analysis begins by identifying the link positions within the individual closed loop, utilising a method outlined by [29]. In this method, each linkage is represented as a vector extending from one joint to another, arranged sequentially from tail to head such that they form a clockwise loop [29].

Once constructed, the set of unknown parameters are identified to define the case of the kinematic analysis. In a closed loop mechanism composed of revolute joints, all link lengths are known, leaving two unknown angles. This arrangement corresponds to case 2c outlined in [29]. The positional equations for this case are defined by equations 7.26 and 7.27 in [29]. Once determined, these equations are differentiated with respect to time to obtain the velocity relationships, which correspond equations 7.30 and 7.31 in [29].

For a closed loop mechanism in the mentioned configuration, these equations can be simplified due to the fixed length of each link. As a result of this configuration, the velocity in the link direction is zero, with each link experiencing an exclusively angular velocity. Following the same differential procedure with the velocity equations, the acceleration relationships may be obtained. These relationships are characterised by equations 7.38 and 7.39 in [29], which can again be simplified to only contain angular accelerations and velocities. Upon the completion of this derivation, equations are obtained to relate the changes in geometric configuration of the mechanism to the positional, velocity and acceleration changes.

2.8 Electronics

The electronic subsystem of a legged robot is vital to its control and operation. The subsystem encapsulates the power system, actuation, environmental perception and decision making of the robot. To ensure an electronic system that suits the scope of this project is selected, a literature review of its components has been conducted.

2.8.1 Microcontrollers

A core component of the electronic subsystem is a microcontroller, or onboard computer. This component is typically used to handle motion control, environment perception, object detection and decision making [26, 30, 31]. The electronics subsystem of quadruped robots often incorporates two microcontrollers when tasks become exceedingly complex, with one controller dedicated to inverse kinematics and gait control whilst the other communicates with the external environment [31]. Common microcontrollers found throughout literature are the Teensy 4.0, ESP32 DevKit and the RasberryPi 4.0 [30, 31].

Teensy 4.0 hosts an ARM Cortex-M7 chip that runs at 600MHz, with 1024 KB of RAM and 2MB of Flash memory [32]. The Teensy board is typically used for motion control only; however, some implementations utilise it as the primary controller [1, 31].

The ESP32 DevKit makes use of 32-bit LX6 dual core processor that runs up to 240MHz, with 520 KB of SRAM and 8MB of Flash memory [33, 34]. What sets ESP32 apart from the Teensy board is its onboard WIFI and Bluetooth module that enables it to connect easily to peripherals [33, 34].

The RasberryPi 4 is the most powerful of the boards and is generally utilised for object detection and environmental perception [31]. It utilises an ARM Cortex-A72 processor with up to 8GB of RAM and an SD slot for memory [35]. This board typically consumes more power and is significantly more expensive than the alternatives provided in literature.

2.8.2 Actuators

Actuation is the sole contributor to the movement in a quadruped system. Through the review of literature, the types of actuators and their advantages used in past robotic systems can be identified.

The first classification of actuators that dominate modern quadrupeds are electrical actuators. These actuators are driven by electrical power, offering a clean, indoor operable and easily controllable form of actuation, which requires no pressurised fluids, fuel tanks, or exhaust systems [36]. Within this class, there consists of three main actuator archetypes: Series Elastic (SE) actuators, Quasi-Direct Drive (QDD) actuators and Direct Drive (DD) actuators. SE actuators integrate compliance between the gearbox and output of the actuator through an elastic element [37]. This configuration allows for highly dynamic movement without damage from impulse contact forces. However, SE actuators can be expensive, bulky, and limited in torque and stiffness control [26, 38]. QDD actuators utilise a low gear ratio transmission in conjunction with a high torque motor [26]. The use of a low gear ratio allows these actuators to achieve back drivability, enabling high torque control and variable nonlinear dampening [26]. The introduction of a transmission can, however, lead to transmission errors, such as backlash, which reduces accuracy [1]. In contrast, DD provides the simplest form of actuation by coupling motors directly to the joints, without the use of gears, belts or chains [39]. This design removes backlash and improves robustness and the mechanical efficiency of the system [39]. Despite these benefits, DD coupling is prone to high thermal temperatures with actuators operating in a high torque low speed band to compensate for the absence of a gear reduction [39].

Other forms of actuation found in previous quadruped designs include hydraulic and pneumatic actuators. These actuators utilise a pressurised liquid or gas to drive joints, providing a quicker response, precise movement, higher power density and increased bandwidth in comparison to electrical actuators [36]. The application for these actuators are, typically, systems designed to carry high payloads [36]. Quadruped robots have been moving away from these actuators in modern times as they require complex heavy systems that can be noisy and are often energy inefficient due to their constant pressure requirements [4, 37].

2.8.3 Power System

Lithium-Ion batteries are the prevailing power source for legged robots, offering high energy density in compact arrangements [40]. Advanced forms of these batteries are commercially available, offering off the shelf components at an affordable price [26].

These types of batteries can, however, become subject to accelerated degradation as a result of extreme temperatures, overcharging and overdischarging [41]. As such, power system should include battery management systems to monitor the battery health in real time via its electrical characteristics [41]. Such a sensor includes Adafruit's INA260 which has been successfully implemented alongside lithium-ion batteries in [42] and [43].

Another component that is often used to regulate the power output of the batteries is a DC-to-DC buck converter [30, 31]. These converters typically reduce the battery voltage to approximately 5V in order to safely power the microcontrollers [31].

2.8.4 Sensors

Sensors form the primary method for quadruped robots to attain information regarding their surrounding environment [9, 11]. There are two groups of sensors commonly applied in legged robots: proprioceptive sensors which measure the internal state of the robot and exteroceptive sensors for measuring external environment information [9].

Typical proprioceptive sensors include inertial measurement units (IMUs), encoders and torque sensors, which are utilised for balance, posture and movement control [9, 11]. Exteroceptive sensors on the other hand, encapsulate visual sensors, such as RGB cameras and non-visual sensors such as LiDAR, which are used for object detection, terrain mapping and navigation [9, 11].

RGB and RGB-D cameras are commonly used in literature to provide object detection and situational awareness [26]. An example of its usage is in [26], where an Intel RealSense D435 RGBD camera provides environmental information required to

traverse rough terrain and for sustained autonomy. To utilise such components correctly, complex control strategies, significant energy sources and additional components such as onboard computers are required.

IMUs provide the ability to estimate the state of the robot without the need for positional information of the actuators or leg mechanisms [31, 44]. IMUs can provide information with up to 9 degrees of freedom, utilising 3-axis acceleration sensors, 3-axis gyroscopic sensors and a 3-axis compass [45]. The noise from hobby grade IMUs can pose a challenge for accurate pose estimation, however, this can be limited by applying various filters [45]. These filters include a moving average filter, Madgwick orientation filter and complimentary filter [31, 33, 45].

2.9 Control System

The control system of a quadruped robot refers to the implementation of complex algorithms to regulate its dynamic motion, position, stability, agility and speed [4]. Unlike fixed-based robotic systems, quadruped motion is governed almost exclusively by the interaction between their feet and the ground [4]. As such, control system algorithms must consider the reaction forces and frictional forces that result from such contact [4]. Throughout literature the implementation of control systems varies in complexity, from simple open-loop control to intricate automation strategies [4, 46].

2.9.1 Open-Loop Control

Open-loop control strategies refer to motion control algorithms that operate without the need for global feedback about the state of the environment through sensor information [46]. Instead, these systems rely upon predefined dynamics to achieve stability in motion, with their success reliant upon robust motion planning [4, 46]. This motion planning typically consists of gait sequencing and trajectory generation [1, 46].

The simplest form of gait sequencing in an open-loop control system is phase offset, or phase shifted control. In this method, each leg performs the same periodic trajectory, whilst the combined motion or gait is defined by phase relationships between each leg

[47]. These relationships reside around the phase difference or offset between leg motions, resulting in alternate timings between swing and support phases of individual limbs [47]. This type of motion control is considered relatively static, allowing minimal corrections to the gait cycle.

To increase stability, a motion control technique known as Central Pattern Generation (CPG) can be utilised. This technique generates stable periodic signals via the simulation of neurons in low-level organisms [4, 44]. This method provides self-stabilisation, determining joint positions and velocities in such a way that the phase relationship between legs becomes easily adjustable [4].

Another variation is the Spring-Loaded Inverted Pendulum (SLIP) model which represents the leg as a massless spring with a load [4]. This model provides precise jumping control of the quadruped robot, however, fails to adjust parameters such as step frequency and cycle, restricting motion to symmetrical gaits [4].

2.9.2 Closed Loop Control

Closed-loop control systems operate by continuously comparing the output of the system with the desired output and adjusting the system to reduce resultant deviations [48]. These control strategies require feedback about the state of the system that is typically obtained from its sensors.

One of the most popular closed-loop algorithms in industry is a PID controller [49]. These controllers apply a manually tuned set of gains: proportional (k_p), integral (k_i) and derivative (k_d) gain, to the error signal of the system in order to optimally control the motion of the robot [49, 50]. The equation for this controller is sourced from [50] and given below:

$$\mu_{PID}(t) = k_p e(t) + k_i \int e(t)dt + k_d \dot{e}(t) \tag{4}$$

An advantage of PID controllers is their ability to be cascaded, such that the output of one controller is fed into the input of another. An example of a cascade PID controller is in [50], where two PID controllers were used for positional and velocity control.

Fuzzy motion controllers have been proven in literature to solve complex control problems [49]. These controllers work through a three-stage process of fuzzification, interference and defuzzification, during which tailored membership functions and interference engines are applied [49]. Each controller typically takes two inputs, an error signal and change in error signal, and produces a single control signal [49]. These controllers greatly improve overall performance, allowing quadrupedal robots to stably traverse set paths, whilst decreasing the control efforts when compared to PID controllers [49].

The final closed loop system considered for this project is the Zero Point Movement (ZMP) model. ZMP works by ensuring that the resultant force vectors experienced by the robot intersects the ground within the robot's support polygon [4]. This control strategy provides excellent performance during low-speed locomotion, however, can pose challenges due to the time-consuming nature of its trajectory planning, which limits response time to disturbances [4].

Chapter 3. Experimental Procedure

Throughout the completion of this project a standardised procedure was applied to the design, analysis and implementation of the individual subcomponents of the system.

The first step of this generalised procedure was to identify existing processes or past examples implemented for the subcomponent in question. This step allowed for the technical knowledge required for this subcomponent to be obtained. Following the collation of past examples, the next step was to evaluate the top performing implementations against the scope of the project.

Once a base implementation was identified as best suited for this project, it could be adapted through an iterative design process. This process was broken into two iterative phases: a theoretical and a practical iteration process. During the first stage a theoretical design was proposed, tested and redesigned until an acceptable configuration was achieved. Next the same three intermediate steps were applied with a physical or practical application of the subcomponent until the final design was achieved.

Finally, the results of implementation and its design process were documented. The following sections outline the design, analysis and implementation for the final versions of the subcomponents only.

3.1 Leg Mechanism Design

3.1.1 Selection of a Suitable Leg Mechanism

The selection of a suitable leg mechanism has a significant influence over a robot's overall performance and its intended function. As such, a literature review was conducted to evaluate mechanisms applied in past implementations against the scope of this project.

Upon the consultation of literature, two possible leg configurations were identified. The first concept model was inspired by ANYmal, implementing a 3 DOF mammal-type serial configuration, with an offset of the HFE and KFE joints. This design was

unique to ANYmal and enabled 360º rotation of the links, giving ANYmal the ability to reconfigure its leg structure and removing the need to consider collisions during the control of the mechanism [11]. The resultant model is depicted in Figure 11(A), illustrating a simplified conceptual arrangement, omitting the implementation of actuators, bearings or key dimensions.

The second concept implements a 2 DOF closed-loop mechanism in the parallel leg configuration. This mechanism is a derivation of the Minitaur robot designed by the University of Pennsylvania, which implements a coaxial five-bar linkage [51]. To remove the design difficulties that arise with the coaxial arrangement, this concept extends the length of the ground link from its original zero length, separating the actuators and obtaining a more conventional linkage [1, 51]. An illustration of this concept is depicted in Figure 11(B) below.

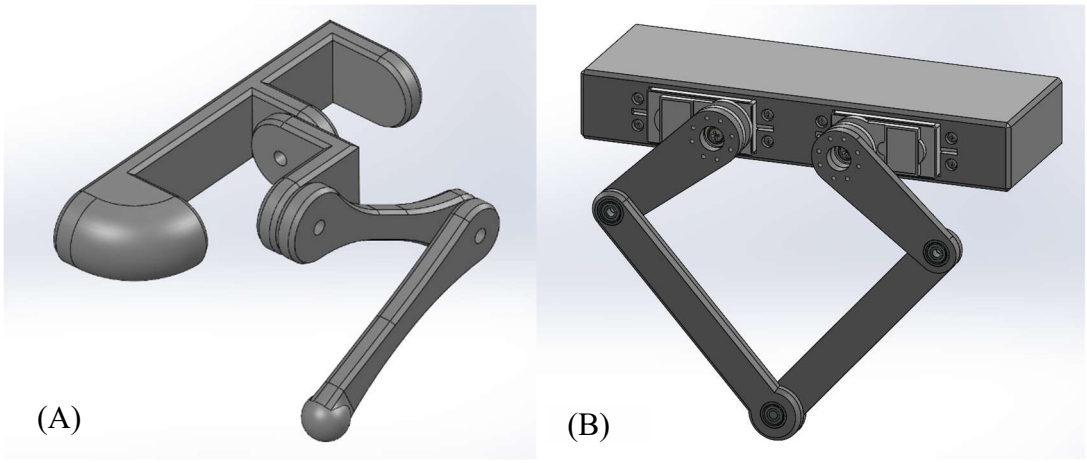

Figure 11: Leg Mechanism Concept Models.

Ultimately, the parallel mechanism was selected as the leg mechanism for this project due to its reduced number of DOF. The five-bar linkage eliminates the need for a HAA joint, typically utilised for steering, by implementing a differential drive system. This, in turn, reduces the total number of actuators required for the platform, thereby reducing the overall weight of the system and simplifying the controller implementation [1]. Along with the reduced DOF, the parallel mechanism provides a

higher carrying capacity for the platform, producing a higher foot force than its serial counterpart [15].

These advantages align with the scope of this project which aims to develop a modular platform that implements a basic control structure and is capable of carrying accessories.

3.1.2 Deriving Parameters of the Selected Leg Mechanism

With a selected leg mechanism in place, the parameters of this mechanism, including the lengths of each linkage, must be defined. To do this, a synthesis method for a five-bar linkage of the 5-RRRRR(a) type outlined in [28] can be applied. This method derives the linkage parameters in such a way that the resulting dexterous workspace of the manipulator is considered singularity-free [28]. In this context, singularities are divided into two categories, type I and type II singularities [28]. Type I singularities are defined as points in space where the actuation of either drive linkages leads to no movement of the end effector, whilst type II singularities refer to a point in space where an infinitesimal number of solutions exist for the positions of the non-actuated linkages [28].

The first step of this method is to define a desired dexterous workspace in which the end effector, in this case the foot of the leg mechanism, is intended to operate. The dexterous workspace is governed by a rectangular geometry defined by a length, width and offset distance from the ground link. This workspace is also considered to be symmetrical about an axis that passes through the origin and is perpendicular to the ground link.

To determine the length, width and offset of the workspace a bounding box was placed around the maximum expected gait trajectory. The maximum expected trajectory was informed by a past implementation in literature, where [1] utilised a step trajectory with a step length equal to twice that of the step height. In this implementation a torso length of 210mm and a height of 64mm was defined [1]. With the intent to prevent overlapping between the workspaces of two adjacent legs in order to avoid collisions and thereby simplifying the required control strategy, a maximum step length of 100mm can be defined. Similarly, to prevent overlap and collisions between the end

effector and torso, a minimum offset from the ground link of 50mm was selected. Consequently, for this project, the maximum step length, maximum step height and minimum end effector offset is defined by 100mm, 50mm and 50mm, corresponding to the length, width and offset of the workspace respectively.

Following the definition of the workspace, two additional parameters must be established: the minimum transmission angle and length of the ground link. For this project, the minimum transmission angle was defined as 20º to reflect the example in [28], whilst the length of the ground link was defined as 50mm, which is equivalent to half the step length to maintain simplicity across the design.

The value of these parameters is summarised in Table 3 below, where M1 to M4 indicate the coordinates of the rectangular workspace:

Table 3: Parameters for the Synthesis of the Selected Leg Mechanism.

Parameter	**Value**
$M1$ (bottom left of workspace)	(-50mm, 50mm)
$M2$ (bottom right of workspace)	(50mm, 50mm)
$M3$ (top right of workspace)	(-50mm, 100mm)
$M4$ (top left of workspace)	(50mm, 100mm)
μ_{min}	20°
l_1	50mm

The resulting configuration of the selected five-bar linkage mechanism can be displayed based on the predetermined parameters. Figure 12 below depicts the geometric arrangement of the leg mechanism including, the individual linkages (l_1 to l_5), the dexterous workspace bounded by points M1 to M4, the minimum transmission angle (μ_{min}), the respective origin and the axis of symmetry for the bounded workspace. It should be noted that the y-coordinates of these points have been inverted to depict the leg mechanism in its physical orientation.

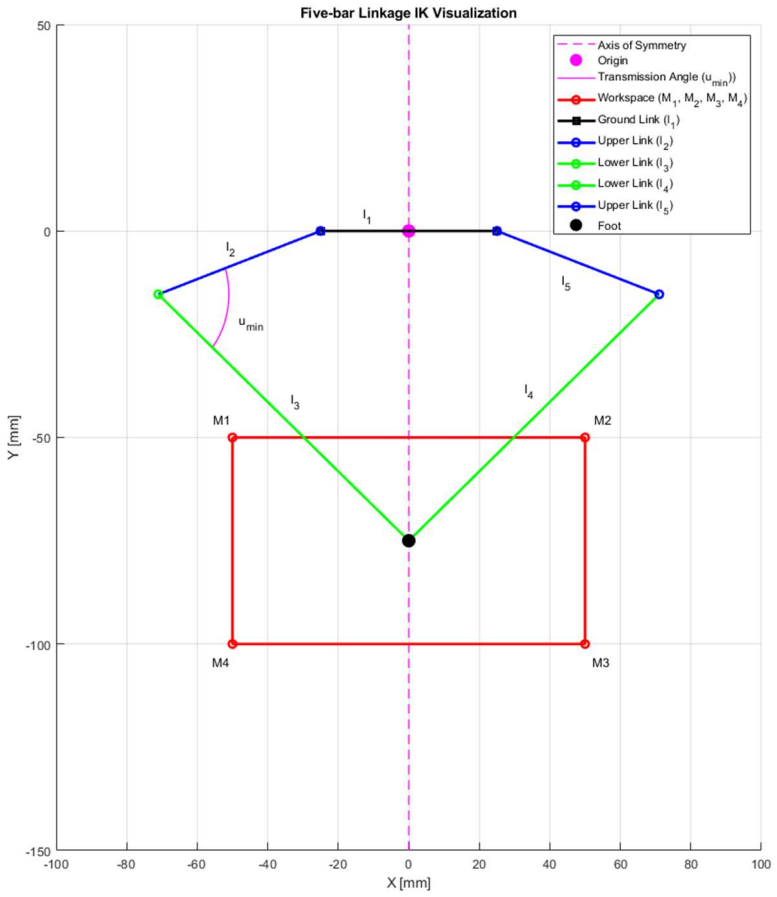

Figure 12: Geometric Representation of Leg Mechanism and Workspace.

Following the determination of these parameters, the synthesis method in [28] applies a system of inequations to derive the remaining lengths of the leg mechanism by imposing a coefficient k. These equations outlined in [28] are given below.

$$l_2 = l_5 = \frac{1}{2}\left(k\sqrt{\left(x_{M3} + \frac{l_1}{2}\right)^2 + y_{M3}^2} - \frac{1}{k}y_{M1}\right) \tag{5}$$

$$l_3 = l_4 = \frac{1}{2}\left(k\sqrt{\left(x_{M3} + \frac{l_1}{2}\right)^2 + y_{M3}^2} + \frac{1}{k}y_{M1}\right) \tag{6}$$

$$k = \sqrt{\frac{y_{M1}^2 - y_{M1}^2(1-\cos(\mu_{min})^2)\sqrt{y_{M1}^4 - [\left(x_{M3}+\frac{l_1}{2}\right)^2 + y_{M3}^2]}}{(1-\cos(\mu_{min}))[\left(x_{M3}+\frac{l_1}{2}\right)^2 + y_{M3}^2]}} \tag{7}$$

The resulting link lengths are given in Table 4 below, with the MATLAB script used to derive these parameters available in Appendix A.

Table 4: Geometries of Five-Bar Mechanism.

Link	Length (mm)
l_1	50
l_2	48.5
l_3	92.8
l_4	92.8
l_5	48.5

3.1.3 Design of the Selected Leg Mechanism

The selected leg mechanism for this project implements a 5-RRRRR five-bar linkage. This mechanism is symmetrical about a central axis, consisting of a ground link, two equal length upper links, as well as two equal length lower links, all of which are connected by a total of five rotational joints. Unlike a conventional linkage, the fixed or ground member of this mechanism represents the fixed distance between the actuators mounted on the body of the project rather than a physical bar linkage. The upper links provide motion for the leg mechanism through the combined independent actuation of the servo motors they attach to. The lower linkages close the loop of this mechanism, connecting the upper members with the foot.

To avoid collisions between the linkages and facilitate compact joint placement, the linkages alternate between the front and back side of the mechanism, forming a set of pairs between opposite upper and lower links. The linkages are held together with M3 bolts and contain 623ZZ bearings to reduce mechanical friction and allow for smooth rotation. The complete linkage assembly is attached to the actuators, which are mounted on a face plate within the actuator housing module. The development of this module will be discussed in a later section.

3.1.3.1 Design of Upper Linkages for the Selected Leg Mechanism

The upper linkages of this mechanism are made up of two variations to allow for compact joint placement and to avoid collisions between the links in the mechanism. Originally these linkages were developed to attach directly to the shaft of the actuators, however, upon testing, the linkages became stripped due to the tolerance limitations of 3D printing and the strength of the material. Instead, the designs have been altered to attach to a metal servo horn.

The two upper linkage variants differ in thickness only, with one design accommodating an offset to position it on the front side of the mechanism. Both designs include a pocket on the back side of the link for the servo horn, with two pass-through holes for M3 bolts to fix it in place. These pass-through holes are also combined with a counterbore for the head of the joining bolts. At the extreme of the linkages another pocket allows for a 623ZZ bearing to be press fit into, with another M3 pass-through hole to allow for a connection to the lower members. Each variant also includes chamfers on their outward facing side but remain completely flat on the face that connects to the lower members. Figure 13 and Figure 14 below depict the back and front side variants of the upper links.

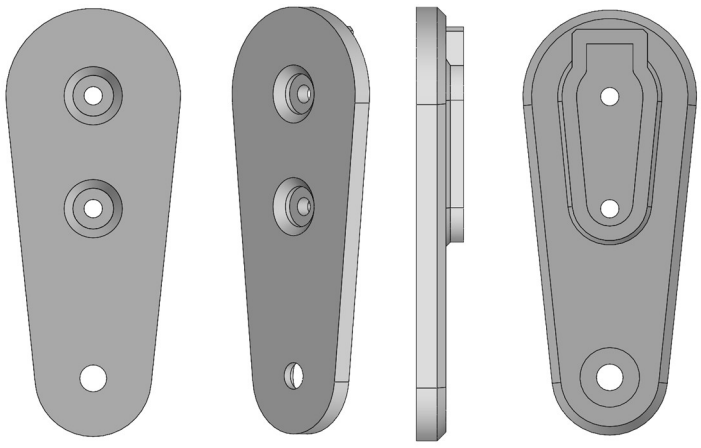

Figure 13: Back-side Variant of the Upper Linkage.

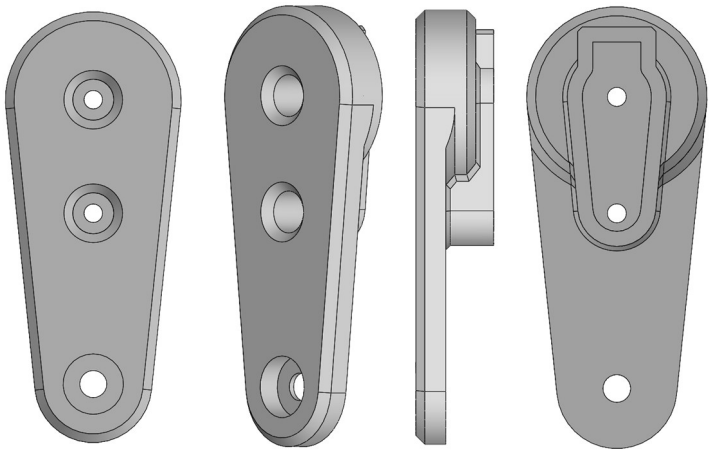

Figure 14: Front-side Variant of the Upper Linkage.

3.1.3.2 Design of Lower Linkages for the Selected Leg Mechanism

The lower linkages in the selected leg mechanism are identical in design for both the front and rear members and are intended to be 3D printed. Each linkage includes two pockets for press fit 623ZZ bearings with accompanying pass-through holes for M3 bolts to allow for a connection to the upper members. The lower end of the linkage is comprised of a half-sphere that forms the foot of the leg mechanism when joined as a lower link pair. The spherical foot extends outward from the edges of the main body of the member, maintain ground contact whilst preventing the remainder of the linkage from colliding with the surface during operation.

A key design feature of the lower linkage is the inclusion of ridges down the front face of the member. This allows the linkage to form an I-beam like structure, reducing the weight of the member, whilst enhancing its resistance to bending stress and impact forces [52]. Much like the upper linkage, this member is completely flat on the face that connects adjacent members to allow for unobstructed rotation. Figure 15 below depicts the final design for the lower link of the selected leg mechanism.

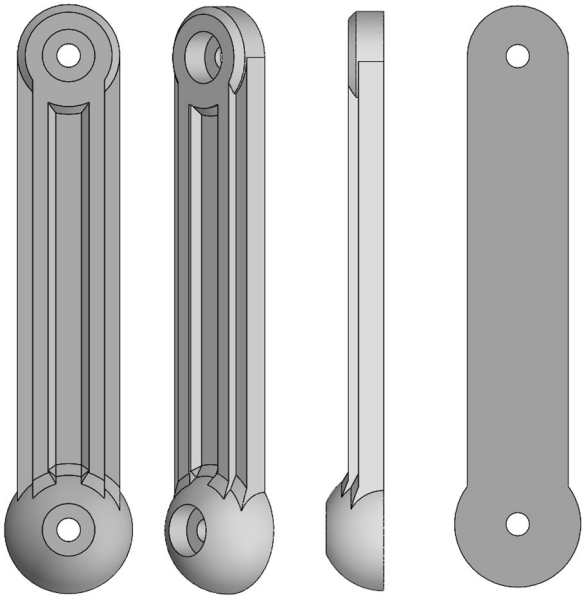

Figure 15: Lower Linkage for the Selected Leg Mechanism.

3.2 Gait Selection and Trajectory Design

The gait design of a quadruped robot plays a critical role in its stability, flexibility and ability to adapt to various terrain [12]. This gait is comprised of two key overarching components, a sequence of joint configurations that define a foot trajectory, and a gait pattern, consisting of multiple out-of-phase foot trajectories for separate leg mechanisms that are combined to provide a continuous motion of the robot. To select a foot trajectory and gait pattern for this project, a literature review was conducted, during which past implementations were evaluated for suitability against the scope of this project.

3.2.1 Foot Trajectory

The trajectory followed by the foot of the selected leg mechanism can be divided into two distinct phases: a swing phase, that generally denotes a curvature path for the lifting of the foot, and a support phase, which describes a horizontal motion whilst in contact the walking surface.

Given that the selected leg mechanism operates with 2-DOF, the trajectory that the foot follows must be described in two-dimensional space. To maintain the simplicity of the control system and align with the scope of this project only basic geometrical paths were considered for the swing phase.

Upon the completion of a literature review, a half-wave sinusoidal trajectory was selected as the swing phase for this project. The half-wave sinusoidal trajectory provided the most mathematically simple implementation when compared to its elliptical and cycloidal counterparts, a critical consideration given the computational power limitations of the microcontroller. An influence in the selection of this trajectory was its adoption in the development of Minitaur and Stanford Doggo, two successful implementations of quadruped robots [15].

The sinusoidal swing phase used in this project can be defined by rearranging Equation 1 from the literature review to form a parametric equation, where t represents the time from zero to the end of the swing phase:

$$\begin{cases} x_m(t) = \pi t \\ y_m(\mathrm{t}) = h \sin\left(\frac{\pi}{x_f - x_s} x_m(t) + \frac{\pi}{2}\right) + y_s \end{cases} \tag{8}$$

For the support phase, a simplistic horizontal line was adopted and transformed to fit the predefined dexterous workspace. The parametric equation for this phase is given below, where t is the time from zero to the end of the support phase:

$$\begin{cases} x_m(t) = \frac{x_f - x_s}{2} - \frac{x_f - x_s}{t} \\ y_m(t) = h + y_s \end{cases} \tag{9}$$

Defining the foot trajectory as a set of continuous functions can present a variety of challenges when attempting to implement them into a practical control system. Instead, to reduce computational complexity, the combined phases can be considered as a single cycle divided into discrete increments. To do this, a total number of equal increments, r, can be defined, with a duty cycle, λ, dedicated for the swing phase.

Given the swing phase represents a half-wave, the phase can be reformulated using a function, $\sigma(d)$, that converts the current increment, d, as a percentage of the resolution duty cycle for the swing phase, λr, to a range between zero and π. This function is defined below:

$$\sigma(d) = \frac{\pi}{\lambda r} d \tag{10}$$

The swing phase can then be altered to include this function, with the final equation for the swing phase is given below:

$$\begin{cases} x_m(d) = \sigma(d) \cdot \frac{x_f - x_s}{2\pi} - \frac{x_f - x_s}{2} \\ y_m(\mathrm{t}) = h \sin\left(\frac{\pi}{x_f - x_s} \cdot x_m(d) + \frac{\pi}{2}\right) + y_s \end{cases} \quad for\ 0 \leq d < \lambda r \tag{11}$$

For the support phase, a separate function, $\phi(d)$, can be applied to convert the current increment, d, as a percentage of the resolution duty cycle for the support phase, $(1-\lambda)r$, to a range between zero and the defined step length $(x_f - x_s)$.

$$\phi(d) = \frac{x_f - x_s}{(1-\lambda) \cdot r}(d - \lambda r) \tag{12}$$

The resulting final equation for the support phase including the discrete interval function is given below:

$$\begin{cases} x_m(t) = \frac{x_f - x_s}{2} - \phi(d) \\ y_m(t) = h + y_s \end{cases} \quad for\ \lambda r \leq d < r \tag{13}$$

By applying these final equations, a single complete cycle of discrete foot positions can be generated. Figure 16 depicts the full cycle trajectory of the foot, whilst Figure

17 illustrates the same trajectory with respect to the predefined workspace. The MATLAB script used to implement these equations and generate the figures is provided in Appendix B.

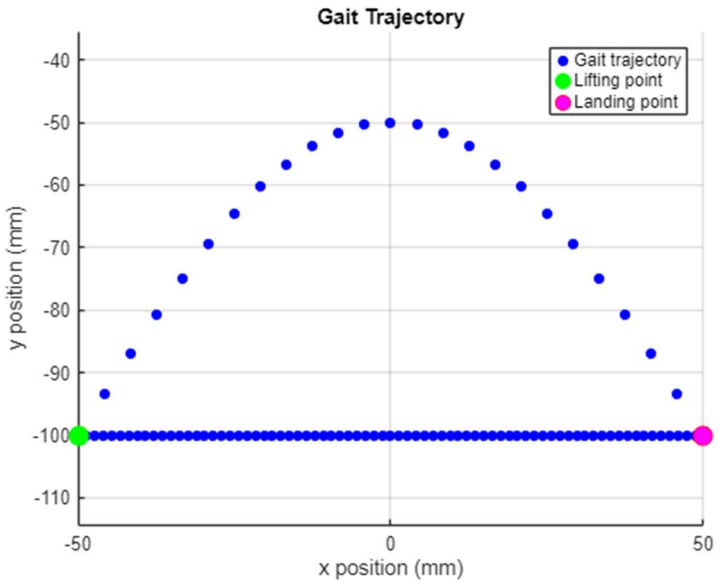

Figure 16: Discrete Cycle of the Foot Trajectory.

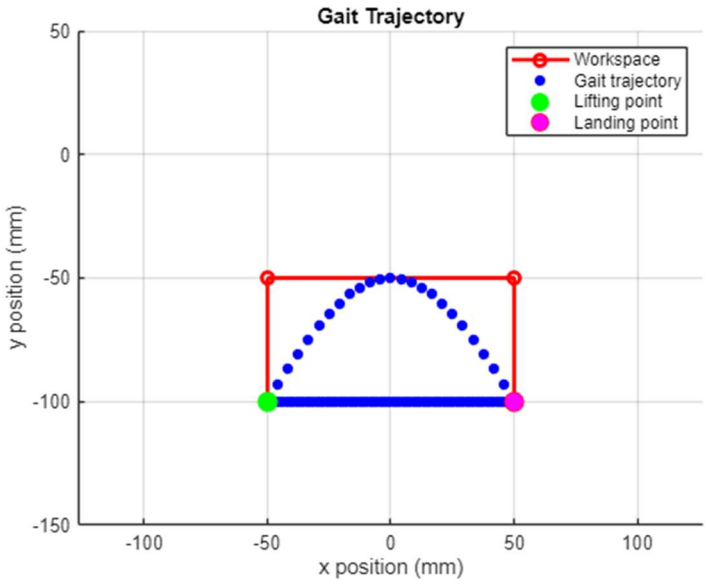

Figure 17: Discrete Cycle of the Foot Trajectory w.r.t the Workspace.

3.2.2 Gait Sequence Pattern

The gait sequence pattern of a quadruped robot defines the coordination between the movement of individual leg mechanisms to provide stable motion of the robot. Upon the completion of a literature review, a variety of coordination patterns were identified as suitable walking mechanisms for quadruped robots.

Initially, a trotting gait pattern was selected for this project, which incorporates an equal-length swing and support phase, with synchronisation between diagonally opposed leg mechanisms. This gait was selected due to its simple implementation requiring two independent controllers instead of four due to the synchronisation of legs. Additionally, trotting gaits support faster movement speeds and lower energy consumption than a walking gait, whilst maintaining greater stability than a bounding or galloping gait [45].

In this configuration, the gait equates to a duty cycle (λ) value of 0.5, with the starting discrete increment offset of each leg depicted as a percentage of the total resolution by Figure 18 below:

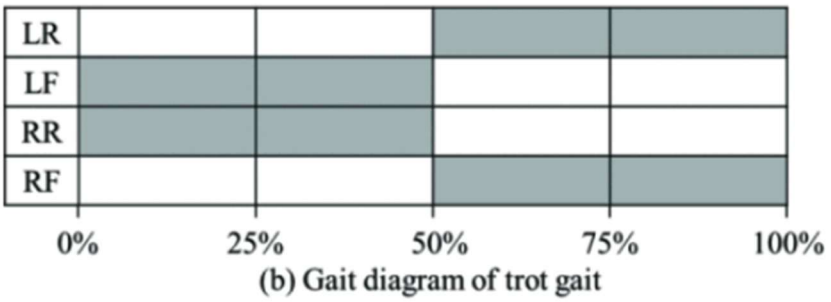

Figure 18: Gait Pattern for a Trotting Gait sourced from [45].

Although this gait pattern was verified via simulation, it became exceedingly unstable in the physical prototype. This instability was brought upon largely due to the reduced speeds of the servo motors, which failed to transition between phases quickly enough resulting in excessive tipping of the robot. Instead, a walking gait was implemented to maintain three points of contact with the ground, improve stability at the cost of faster

movement speed. The transition to a walking gait was completed through an update of the λ value to 0.25 and a change in the starting discrete increment for each leg. The resulting sequence for a walking gait is depicted in Figure 19 below.

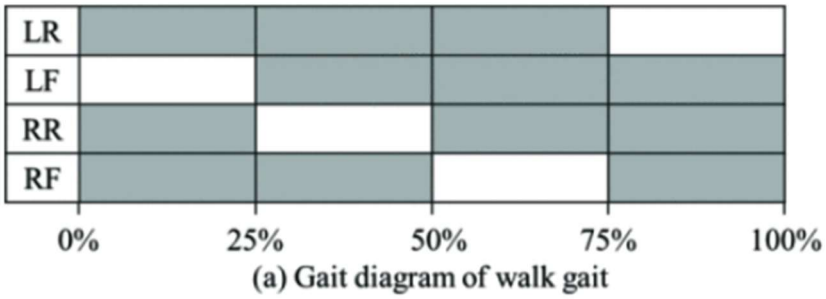

Figure 19: Gait Pattern for a Trotting Gait sourced from [45].

3.3 Design for Modularity

A key limitation of current quadruped robots arises from the degradation experienced by mechanical components such as leg joints and actuators, as well as the volatility of electrical systems when exposed to water, dust and extreme temperatures [2, 9]. The frequent maintenance that results from such issues, complicates the deployment of quadruped robots, requiring routine field checks and replacement parts to be considered in operational planning, driving up cost and down time [2].

In addition to frequent maintenance requirements, the review of literature has identified the need for future quadruped robots to design for multifunctionality and adaptability [10]. To address these challenges and build towards a robotic platform for hybridisation, modularity is introduced into the development of this project.

By designing modular components, parts can be readily replaced or upgraded, improving the overall adaptability of the robot's function and increasing the ease of maintenance [2, 53]. Moreover, modularity reduces training requirements for maintenance staff, streamlines repair and promotes reconfigurability of the robot [53].

The foundation of modularity in this project consists of two distinct stages: component modularity and operational modularity. Component modularity aims to standardise the mechanical framework that operation modules are built upon, whilst operational modularity endeavours to separate the subsystems of the robot into individual modules.

3.3.1 Component Modularity

To improve consistency of the robot's structure and aid reconfigurability, the mechanical framework of the operational modules in this project is constructed from a set of standardised components. Each operational module consists of one or more cubic units connected in a lattice-like arrangement, which form the foundational elements of the mechanical structure.

Each volumetric unit measures 85mm x 85mm x 85mm and is divided into two fundamental components: a wire frame and face plates. The dimensions of this unit were determined through the side-by-side connection of two units, housing a single actuator each, where the resultant operational module completely contained the leg mechanism. The units were then scaled to a size that in the event two leg modules (four units) are joined, the mechanisms do not collide. This is illustrated by Figure 20 below.

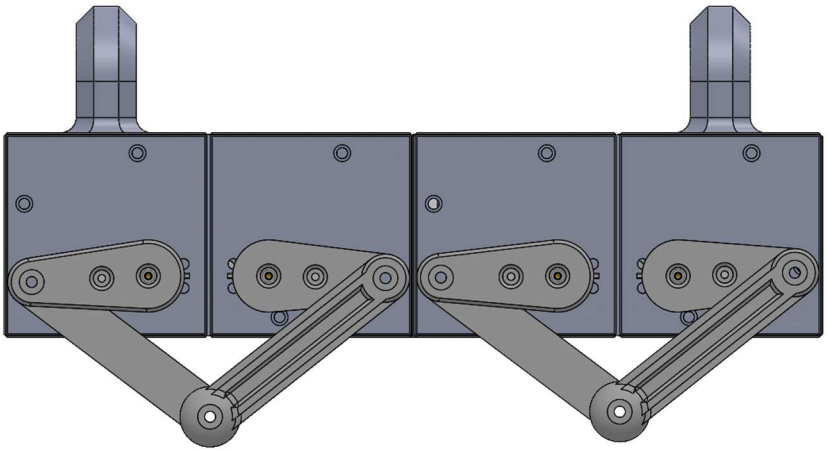

Figure 20: Configuration of Two Adjacent Actuator Modules.

The cubic geometry of these units was selected to allow for the formation of a close-packed lattice, which enables passive self-alignment during assembly and simplifies the required mechanical structure [19]. These characteristics are achieved by the identical planar faces of the cubic unit, which prevents misalignment that may occur in shapes without uniform widths, depths and heights [19].

3.3.1.1 Wire Frame Module

The wire frame serves as the three-dimensional structure that facilitates the connection of all face plate variations to form a foundational unit. Each face of the frame includes four holes for heated inserts utilised to secure face plates via M3 bolts. These holes are offset from the centre of each edge along the face to prevent bolts on adjacent faces from intersecting.

To reduce the total number of require parts for the robot, the wire frames of each unit can be combined in a 3D model via a protruding ridge. This 45º ridge assists with the alignment of face plates on the frame during assembly. An example of two connected frames is depicted in Figure 21 below.

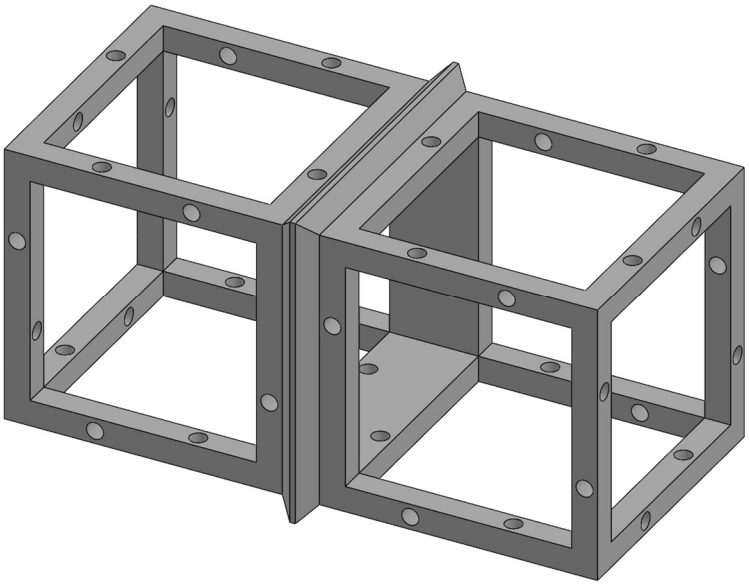

Figure 21: Wire Frame Joining Two Units for Use in the Spacer Module.

By allowing the joining of wire frames, redundant faces and connectors are eliminated and the internal capacity of operational modules is increased. Although this may compromise standardised sizing between operational modules composed of varying numbers of units in different lattice configurations, it reduces the number of parts and fasteners required. With reduced components, manufacturing and assembly time is shortened, minimising the cumulative cost for the entire robotic platform.

3.3.1.2 Blank Plate Module and its Functional Variants

This project includes several variations of the available face plates for each module. The base design for these variations is a blank plate, which acts as a placeholder and template for functional variants. The plate incorporates edges angled at 45º to align with wire frame and eliminate collisions with perpendicular plates. Four holes and counterbores, offset to the right, are also included in this design to align with the wire frame and attach via M3 Bolts. A front and side view of a blank plate is depicted by Figure 22 below.

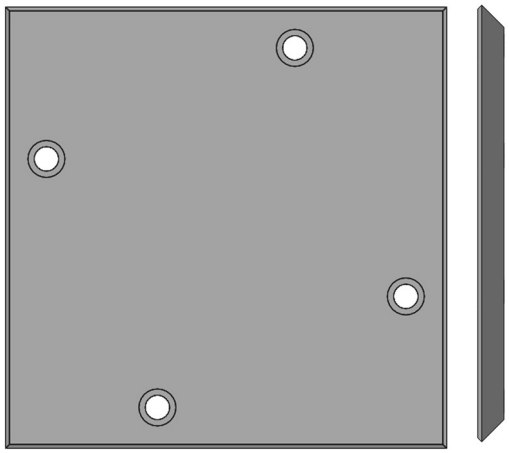

Figure 22: Front and Side View of Blank Plate Module.

Varients of this plate include a logo plate, handle plate, actuator plate, air plate, power plate and electronics plate. The logo variant embosses the MORPH logo, whilst the handle plate includes mounting holes for handles, with the features of each variant placed at the centre of the plate. The air plate variant implements a hexagonal grid to promote airflow, whilst the power plate adds mounting holes for the power switch, charger and aerial. The logo, handle, air and power plates are depicted in Figure 23, Figure 24, Figure 25 and Figure 26 respectively.

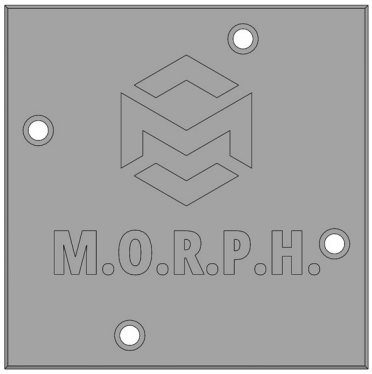

Figure 23: Logo Plate Variant.

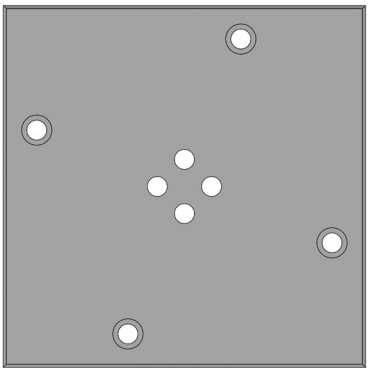

Figure 24: Handle Plate Variant.

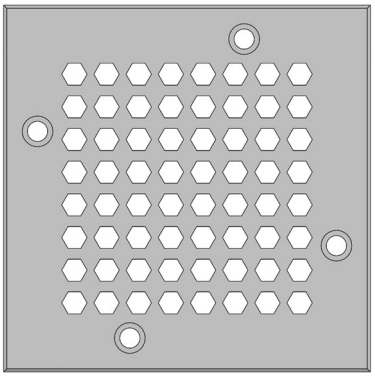

Figure 25: Air Plate Variant.

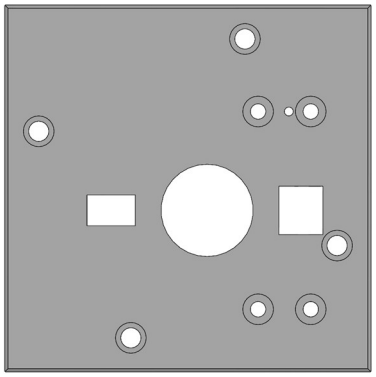

Figure 26: Power Plate Variant.

The actuator plate and electronics plate combine multiple blank plate templates, providing cutouts for their respective components. Combining multiple templates allows these variants to span entire operation modules, reducing the number of parts and fasteners required for these subsystems. These variants are depicted below in Figure 27 and Figure 28.

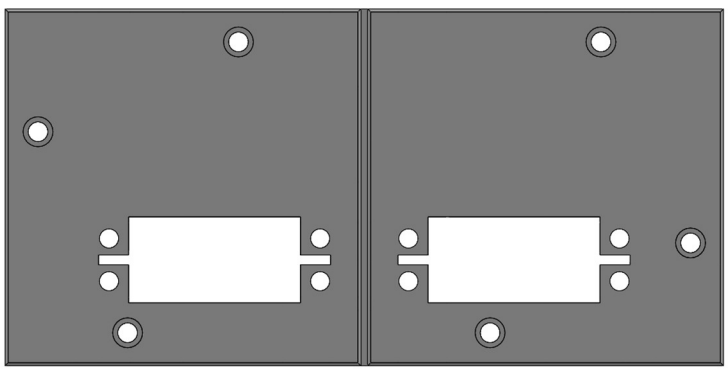

Figure 27: Actuator Plate Variant Combining Two Template Plates.

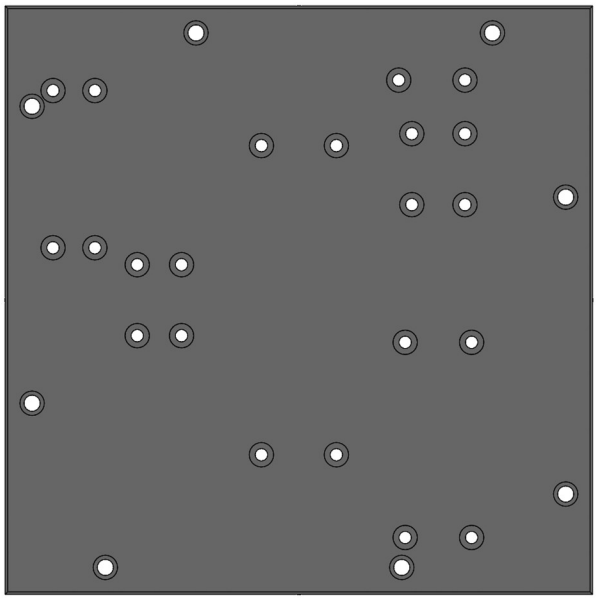

Figure 28: Electronics Plate Variant Combining Four Template Plates.

3.3.1.3 Connection Plate Assembly

Much like other plate variations, the connection plate extends the blank plate template and is designed to mechanically join operational modules, whilst enabling the transfer of both power and data signals between modules.

From a mechanical standpoint, the connection plate implements two distinct joining mechanisms: permanent magnets and bi-gendered mechanical pins. These mechanisms were selected as they offer rapid and simple assembly, self-alignment and high connection strength [25]. To simplify the power requirements and the meet the need for frequent reconnection during maintenance, active or permanent fastening methods such as vacuums, actuators, adhesives and fasteners were omitted [25].

For the permanent magnet portion of the connector assembly, four neodymium magnets are press fit into the corners of the plate. These magnets are arranged such that diagonally opposite magnets share identical pole orientation whilst adjacent magnets are reversed.

To compliment the magnets, passive mechanical pins are implemented to increase the connectors strength relative to shear forces which are a known weakness of magnets.

The passive mechanical mechanism includes four bi-gendered pins (2 male, 2 female), which provides an interference fit that holds two connecting plates together via friction. Similar to the permanent magnet arrangement, diagonally opposite pins share the same gender, whilst adjacent pins are reversed. To allow for ease of manufacturing, male pins are printed separately and glued into place.

The resulting connection plate which leaves room for a custom electrical connector held in place by a twist lock mechanism is depicted in Figure 29 below.

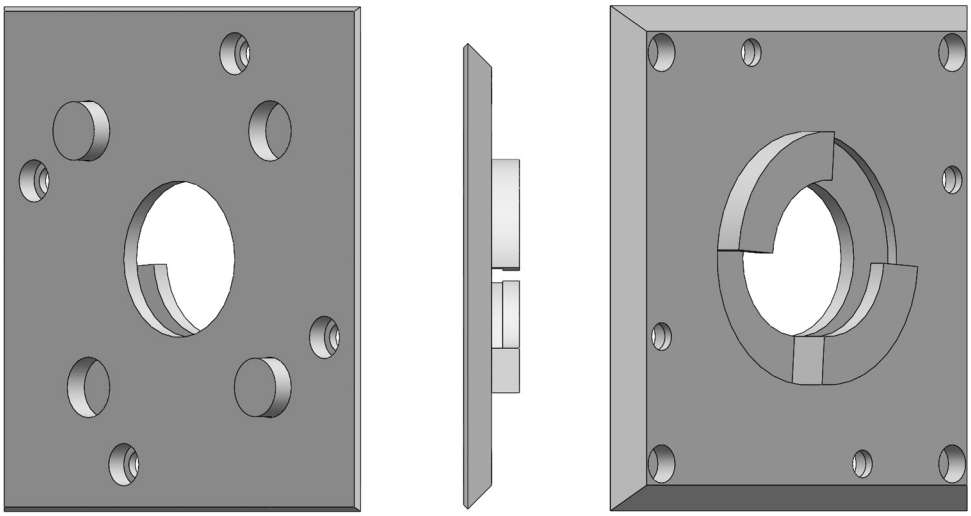

Figure 29: Connection Plate.

As mentioned above, a custom electrical connector was developed to transmit signals between modules. This connector fits into a cutout in the connection plate and utilises a twist lock mechanism to fix itself in place. This twist mechanism is a simplified adaption of the mechanical connectors developed in [54] and [55], and is a derivative of lock and key mechanisms [24].

For the electrical connection, header pins commonly found on printed circuit boards were selected following their success in [56]. Four sets of 3x1 bi-gendered header pins (2 male, 2 female), were arranged in a diamond formation at the centre of the connector, with each set of pins fixed in place with adhesive. Once again, these pins were arranged such that diagonally opposite pins share the same gender whilst adjacent pins are reversed.

In total, 12 individual pins were implemented to allow a full range of signals, utilised in the control of the robot, to pass between modules. Since the connection assembly is designed to be mirrored along a vertical axis, only six distinct signals are available: power, ground, motor 1 PWM, motor 2 PWM, I^2C data line (SDA) and an I^2C clock line (SCL).

To facilitate manual assembly of the connector using the twist lock mechanism, knurling was also added to the rear for grip. The finalised connector is illustrated below in Figure 30.

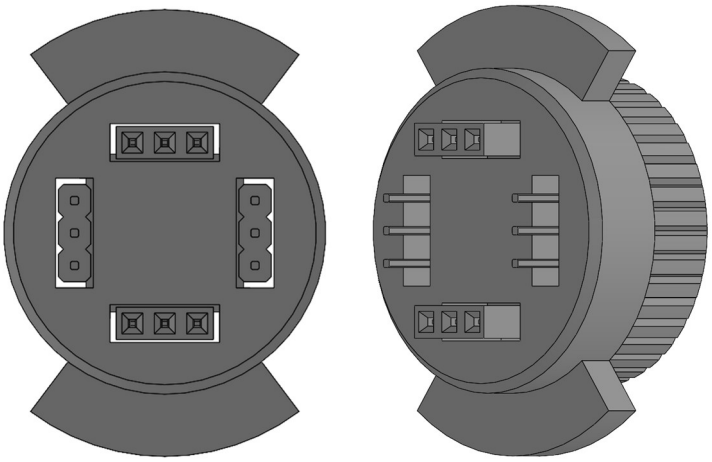

Figure 30: Custom Electrical Connector.

The symmetry of bi-gendered connectors about the diagonals is critical to the design of this assembly. This symmetry enables a single standardised connector to be used for both mating ends of the connection [24, 55]. When two connection assemblies are mated, the interface becomes mirrored about a vertical axis, enabling each male connector to align with its opposing female connector [24, 55]. Without the diagonal symmetry, two connectors of the same gender will collide resulting in a failed connection. It should also be noted that the connector is in fact symmetrical about the horizontal axis, however a connection like this would lead to a mismatch between signal types.

The complete connector assembly, combining the mechanical connection plate and custom electrical connector, is displayed below in Figure 31.

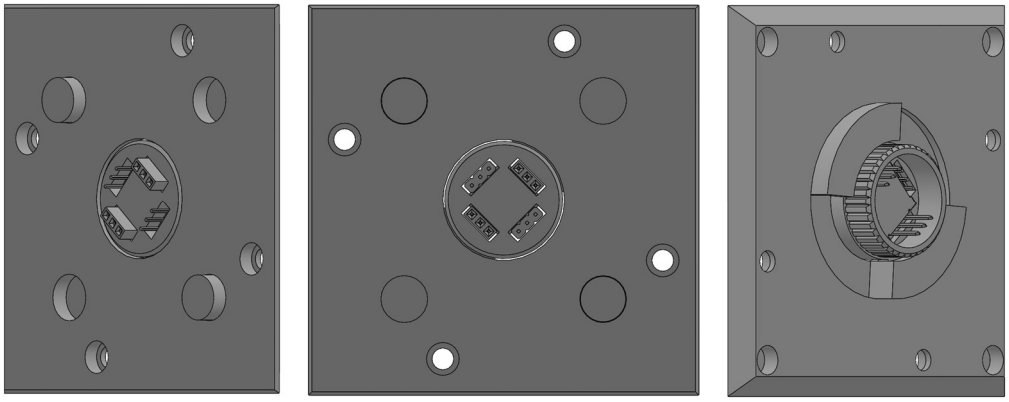

Figure 31: Complete Connector Assembly.

3.3.2 Operational Modularity

Operational modularity is implemented to separate the operational subsystems of the legged robot. For this project, modules have been divided into an electronics module, actuator module and spacer modules.

The electrical module is made of four standardised units in a two-by-two lattice configuration. It houses the electronic subsystem, including the microcontroller, servo driver, power regulator, power source and sensors. The module implements eight connection plate assemblies around its sides, with three air plates and a power plate on top as well as an electronics plate on the bottom. An illustration of this module is given in Figure 32 below.

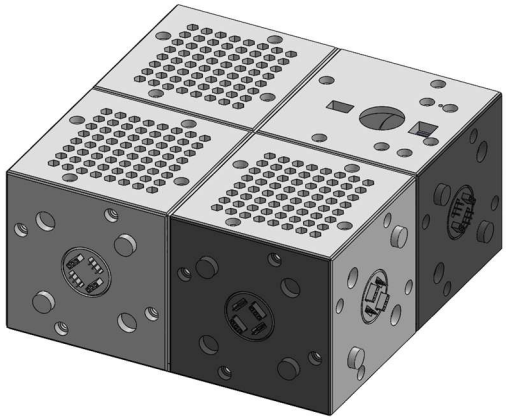

Figure 32: Electronics Module.

The actuator module consists of a two-by-one lattice of standardised units. It contains the servo motors and selected leg mechanism, with the motors attached directly to the actuator plate. The module integrates four connection plates on the remaining sides with blank plates on the top and bottom faces. The resulting configuration is depicted in Figure 33 below.

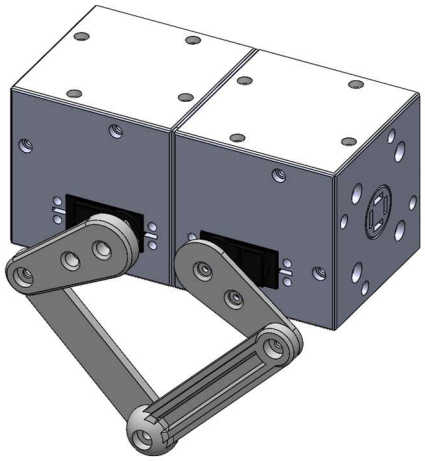

Figure 33: Actuator Module.

The spacer module acts as a filler element to complete the close-packed lattice, allowing for various configurations such as a square to exist without gaps in the lattice. It also provides a surface for additional tooling such as handles to be placed. The spacer incorporates a two-by-one lattice of standardised units, with two logo plates, two handle plates, two blank plates and four connection plates. A depiction of this module is given below in Figure 34.

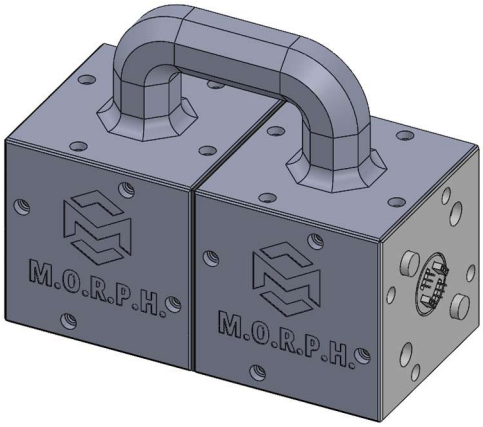

Figure 34: Spacer Module.

3.4 Model of Complete Quadruped Robot

Reconfigurability is a key design principle for this project, enabling the robotic platform to adapt to multiple functionalities. The complete quadruped robot is constructed by combining the described operational modules into a variety of footprints. For the purpose of this project, two key configurations have been selected to showcase its adaptability: a square configuration and rectangular configuration.

The square configuration of the robotic platform implements a four-by-four lattice of standardised units, combining two spacer modules, an electronics module and four actuator modules. This arrangement is representative of a platform that implements a wide base for increased stability, enabling it to complete functions such as hauling large loads. A depiction of this arrangement is shown in Figure 35 below.

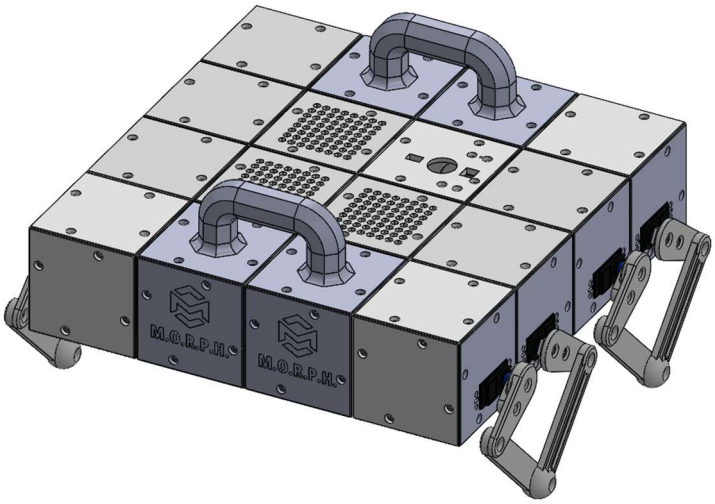

Figure 35: Square Configuration of Complete Robot.

A two-by-six lattice of standardised units are joined to construct the rectangular configuration of this quadruped. In this configuration, four actuator modules are connected such that two modules are positioned at the front and rear of the robot. These modules are then tied together by an electronics module in the centre of the platform. In this configuration, the robot is able to complete functions that require a narrow footprint such as inspections between machinery at a factory. The rectangular configuration is illustrated below by Figure 36.

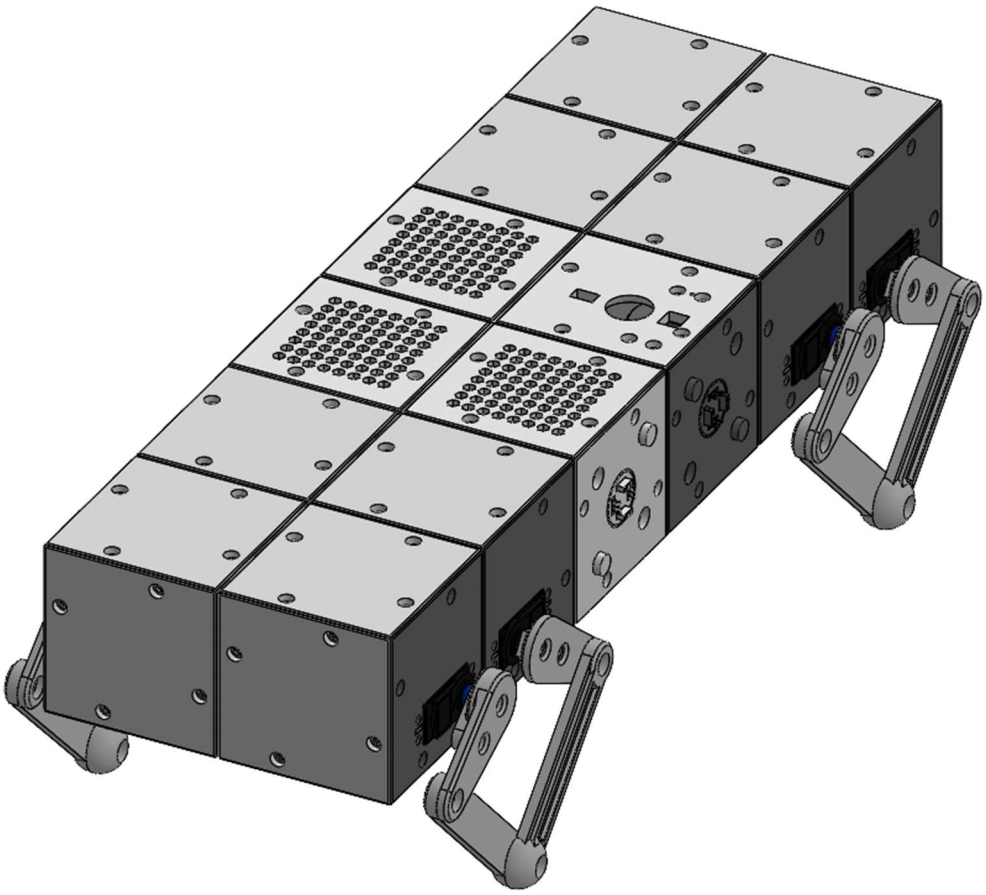

Figure 36: Rectangle Configuration of Complete Robot.

Together these configurations showcase the reconfigurability and adaptability of the platform into a number of different footprints. This is enabled by the modular connectors that allow for rapid reassembly, easier maintenance and the ability to combine additional actuator modules, not constraining it to the current quadrupedal implementation.

3.5 Material Selection

To allow for rapid manufacturing via 3D printing, the predominant material for all components is Polylactic Acid (PLA). This material has been implemented in a wide variety of past implementations, including in [26]. The material properties of PLA, sourced from [57], [58] and [59], are provided in Table 5 below.

Table 5: Material Properties of PLA.

Parameter	Value
Elastic Modulus	3500 MPa
Poisson's Ratio	0.36
Shear Modulus	1278 MPa
Mass Density	1.252 g/cm^3
Tensile Strength	59 MPa
Material Dampening Ratio	0.02
Compressive Strength	62 MPa
Yield Strength	70 MPa
Thermal Expansion Coefficient	0.000076 µm/ ºC
Thermal Conductivity	0.111 W/m·ºC
Specific Heat	1590 J/kg·ºC

3.6 Static Analysis

The static analysis for this project aims to determine the internal stresses and strains, as well as the external moments and torques that act on individual components of the selected leg mechanism as a result of external loading. This static analysis is conducted for three primary purposes: To determine the peak loads experienced by the mechanism, which are pivotal in the process of actuator selection; to validate the mechanical structure of the mechanisms and to verify the material selection. For simplification, the static analysis focuses predominantly on the leg mechanism, providing a representation of the project's overall performance.

3.6.1 SolidWorks Finite Element Analysis

The determination of internal stresses and strains experienced by the leg mechanism is an important step in the verification of the mechanisms structure and material. To achieve this an approach taken in [26], which performs a finite element analysis (FEA) in SolidWorks of the robot in its standing posture, can be applied.

To replicate this approach, a static study was developed in SolidWorks for the following components: the upper and lower linkage, the leg mechanism and a simplified model of the complete robot. For each study, a load equivalent to eight body weights and a material of polylactic acid (PLA) were applied to each component.

3.6.1.1 Finite Element Analysis of the Lower Linkage

The static analysis for the lower linkage component was configured with a load of 35N applied perpendicular to the foot, with the opposing end fixed in place around the bearing pocket.

The results of the analysis indicated a maximum stress, equivalent strain and deflection of 5.962 MPa, 0.0014, and 0.64 mm respectively, remaining almost a magnitude under the materials yield strength. The results of this analysis are displayed in Figure 37, Figure 38 and Figure 39.

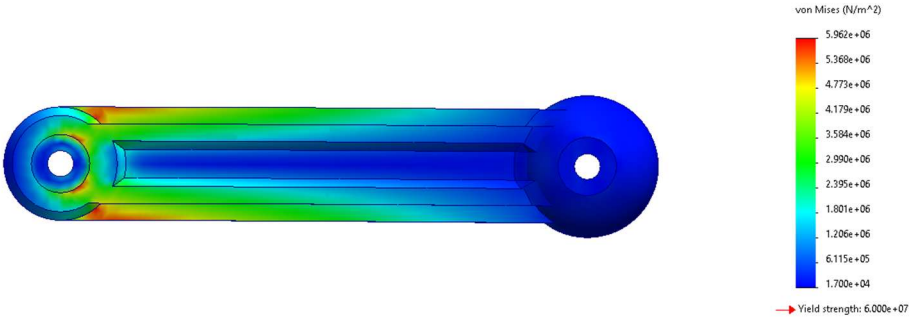

Figure 37: von Mises Stress of the Lower Linkage.

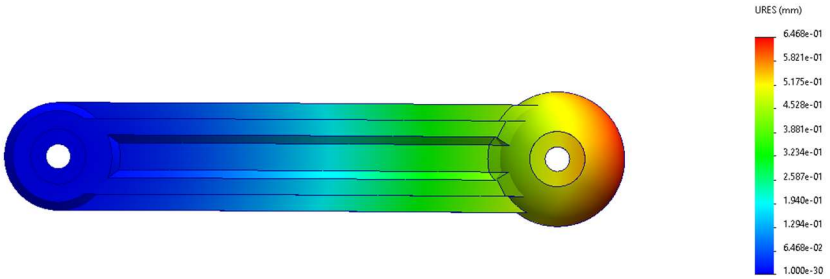

Figure 38: Displacement of the Lower Linkage.

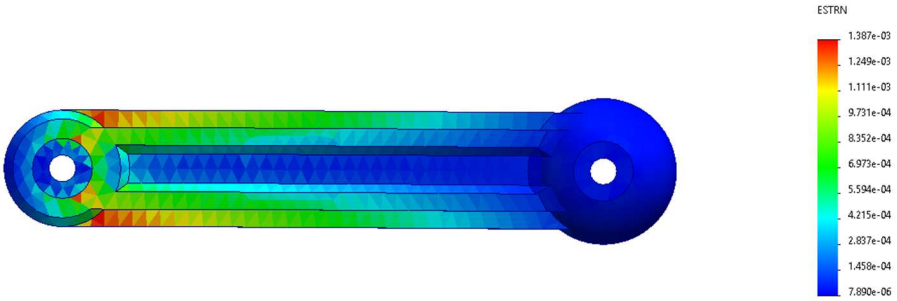

Figure 39: Equivalent Strain of the Lower Linkage.

3.6.1.2 Finite Element Analysis of the Upper Linkage

The static analysis for the upper linkage component was configured with a load of 35N applied perpendicular to the connection point with the lower linkage, whilst the opposing end was fixed in place around the servo horn pocket.

The results of the analysis indicated a maximum stress, equivalent strain and deflection of 3.163 MPa, 0.0007, and 0.04 mm respectively, remaining more than a magnitude under the materials yield strength. The results of this analysis are displayed in Figure 40, Figure 41 and Figure 42.

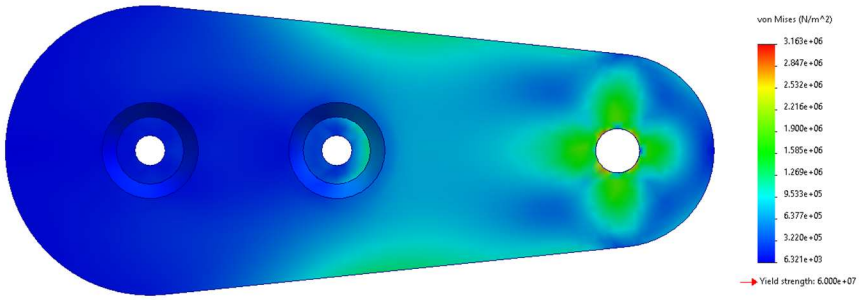

Figure 40: von Mises Stress of the Upper Linkage.

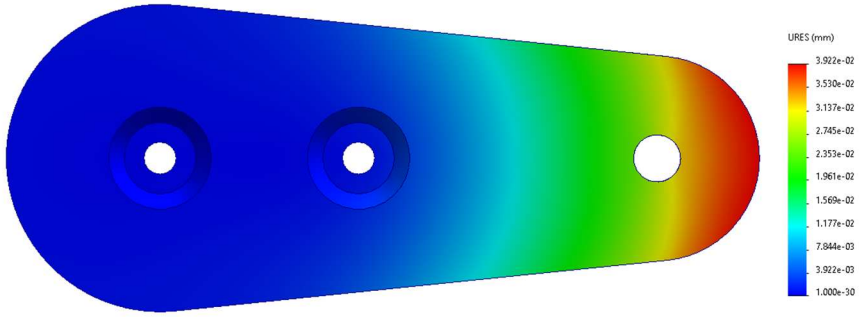

Figure 41: Displacement of the Upper Linkage.

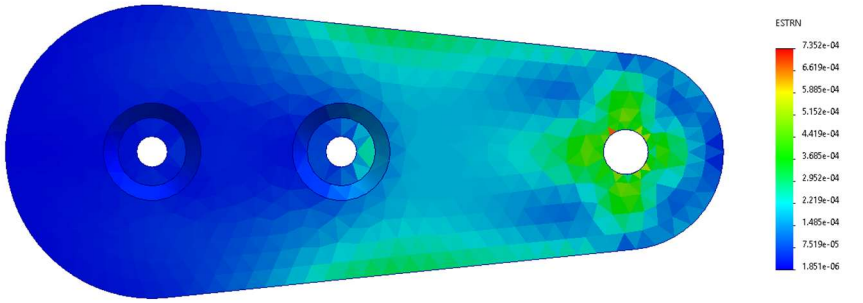

Figure 42: Equivalent Strain of the Upper Linkage.

3.6.1.3 Finite Element Analysis of the Leg Mechanism Assembly

The static analysis for the leg mechanism assembly was configured with two loads of 35N applied vertically to each foot component, with the top plates of the module fixed in place.

The results of the analysis indicated a maximum stress, equivalent strain and deflection of 0.611 MPa, 0.00009, and 0.01 mm respectively, remaining more than a magnitude under the materials yield strength. The results of this analysis are displayed in Figure 43, Figure 44 and Figure 45.

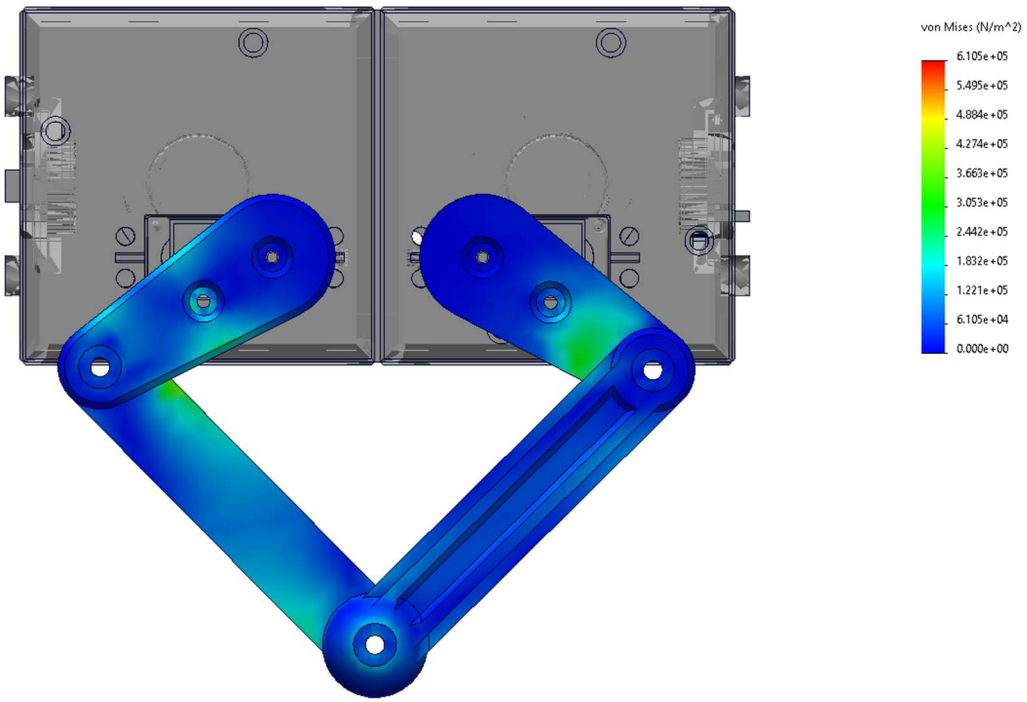

Figure 43: von Mises Stress of the Leg Mechanism Assembly.

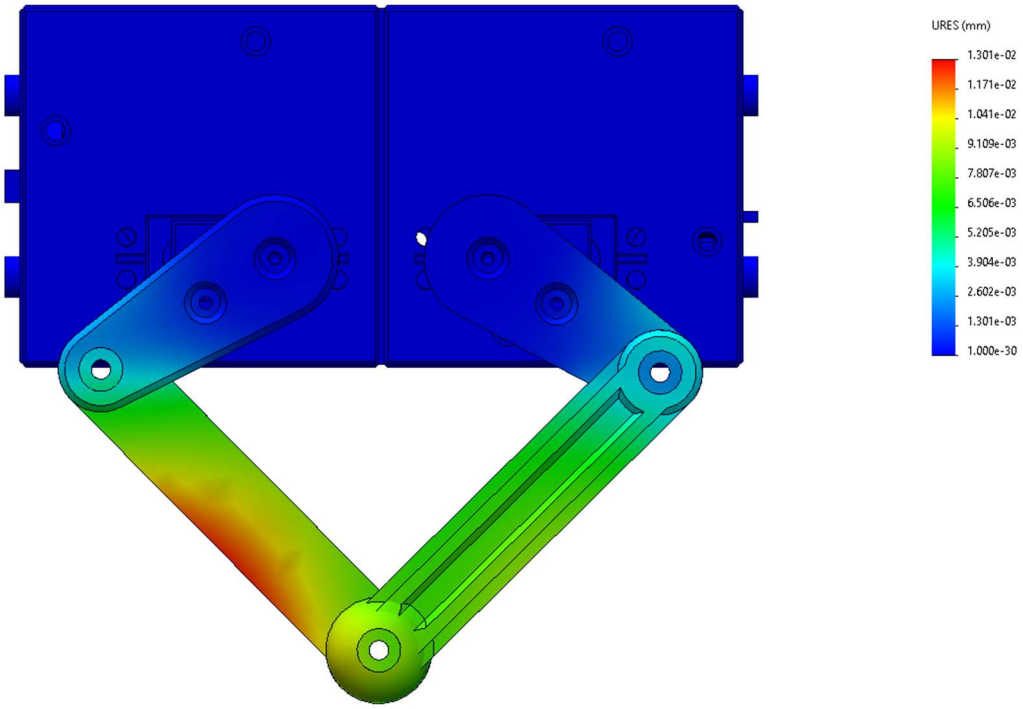

Figure 44: Displacement of the Leg Mechanism Assembly.

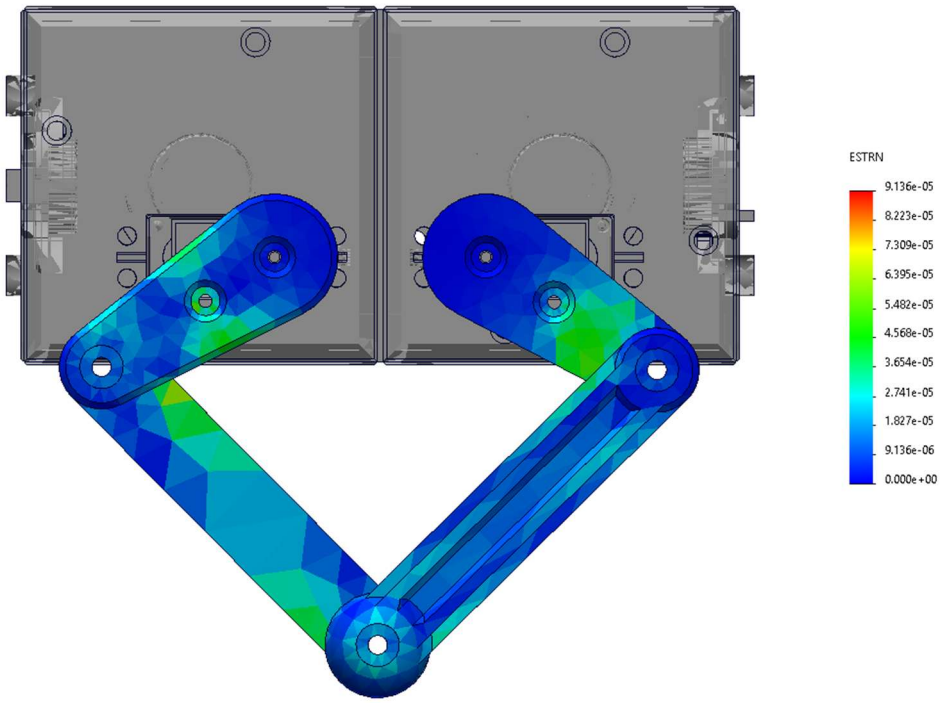

Figure 45: Equivalent Strain of the Leg Mechanism Assembly.

3.6.1.4 Finite Element Analysis of a Simplified Robot Configuration

To reduce computational load of the static analysis, a simplified model of the robot was developed, combining the modules into a singular body reduce complexity. The static analysis for the complete robot was then simulated by applying eight loads of 35N applied vertically to each foot component, with the top face of the body in place.

The results of the analysis indicated a maximum stress, equivalent strain and deflection of 0.611 MPa, 0.00009, and 0.01 mm respectively, remaining more than a magnitude under the materials yield strength. The results of this analysis are displayed in Figure 46, Figure 47 and Figure 48.

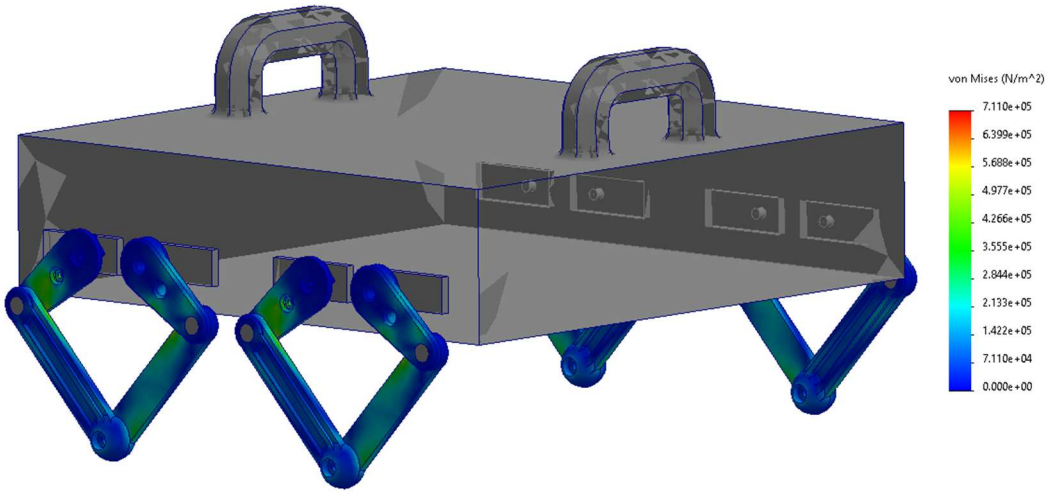

Figure 46: von Mises Stress of the Simplified Robot Configuration.

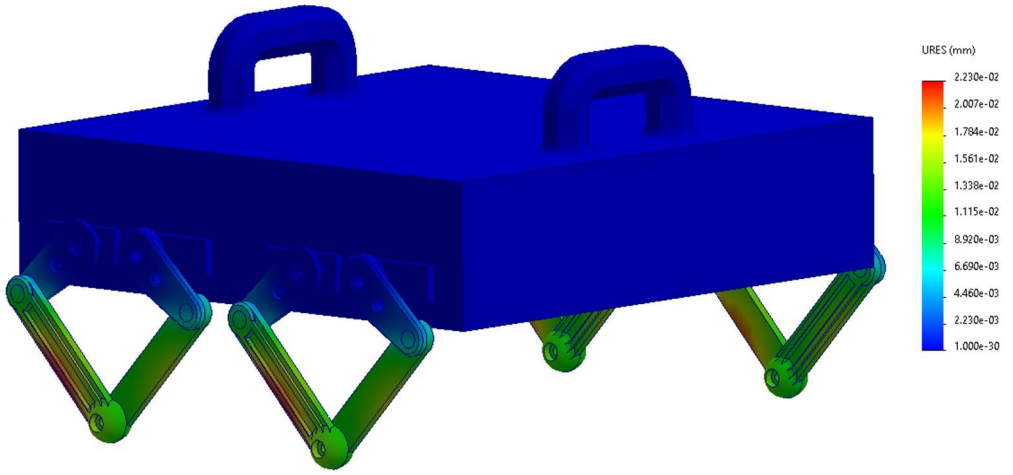

Figure 47: Displacement of the Simplified Robot Configuration.

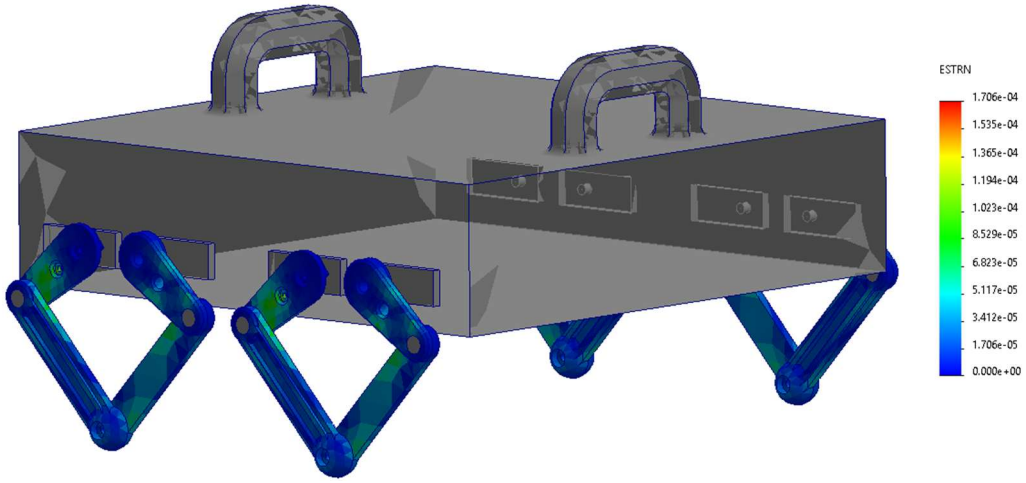

Figure 48: Equivalent Strain of the Simplified Robot Configuration.

3.6.1.5 Summary of the Finite Element Analysis

The results of the static studies validate the mechanical structure and material selection of the robot. In all cases, the observed internal loads remained more than an order of magnitude below the yield strength of the material, even when subject to loads well above the anticipated capacity. These results allow for a substantial safety factor to be applied to the robot's mechanical structure, reducing the likelihood of failure during normal operations.

3.6.2 Free Body Diagram Analysis

To complement the FEA analysis, a static analysis of the mechanism in an arbitrary orientation was conducted to determine the static articulated torque and contact forces experienced by the body. The analysis used in this project, simplifies the approach discussed in the literature review and implemented by [27]. The approach is simplified by considering a single leg mechanism rather than the entire robot.

In this case, the applied contact force is equal to the weight of the mechanism and an additional applied force. This external load is equivalent to one body weight applied over the entire robot or a quarter of a body weight applied to a single leg mechanism.

In this analysis, it is assumed that the reaction force acts perpendicular to the body of the robot which is parallel to the ground and considered fixed. A static frictional force is also resolved onto the foot, acting perpendicular to the reaction force. Applying these assumptions, a static equivalent model can be developed to solve the geometries of the system. This model is depicted in Figure 49 below.

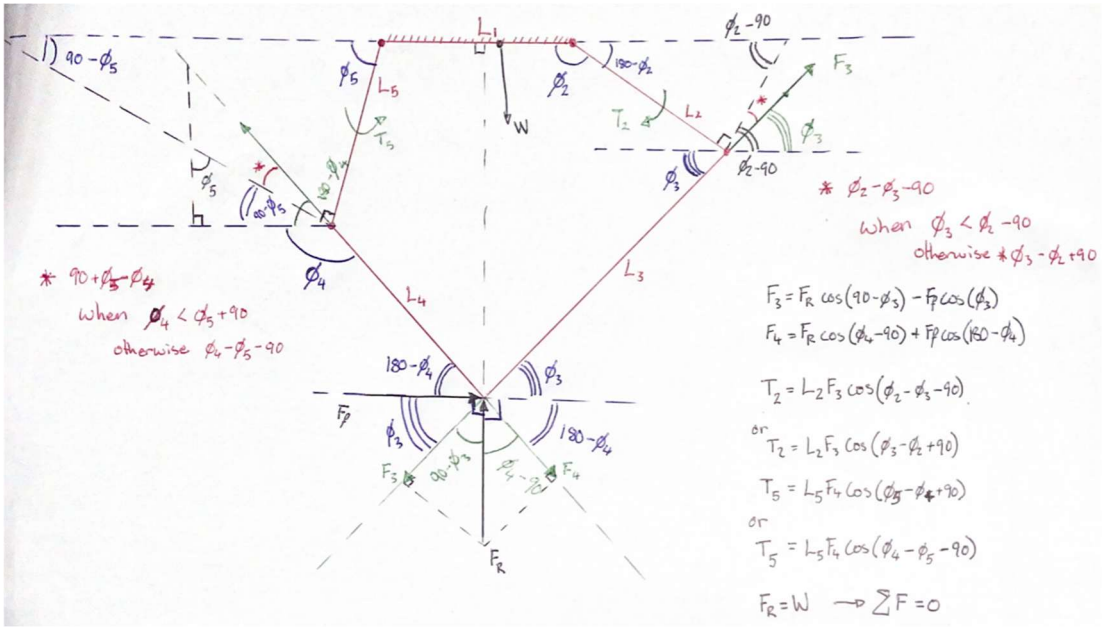

Figure 49: Static Equivalence Model of Selected Leg Mechanism.

Using the geometries of the system, the reaction and frictional forces can be resolved into components that act upon links three and four. Assuming these links maintain equilibrium, a counteracting force is applied to the other end of each link. This force can then be resolved into its components perpendicular to links two and five in order to determine the torques applied to these respective links. The equations for the torques and forces acting upon these links are as follows, where is equal to the coefficient of friction of the contacting surface. If the leg mechanism is not in contact with a surface, this coefficient becomes negligent.

$$F_f = \begin{cases} \mu_s F_R & for\ \mu_s > 0 \\ 0 & Otherwise \end{cases} \tag{14}$$

$$F_3 = F_R \cos(90 - \varphi_3) - F_f \cos(\varphi_3) \tag{15}$$

$$F_4 = F_R \cos(\varphi_4 - 90) + F_f \cos(180 - \varphi_4) \tag{16}$$

$$\begin{cases} T_2 = L_2 F_3 \cos(\varphi_2 - \varphi_3 - 90), & \varphi_3 < (\varphi_2 - \pi) \\ T_2 = L_2 F_3 \cos(\varphi_3 - \varphi_2 + 90), & \varphi_3 \geq (\varphi_2 - \pi) \end{cases} \tag{17}$$

$$\begin{cases} T_5 = T_2 = L_5 F_4 \cos(\varphi_5 - \varphi_4 + 90), & \varphi_4 < (\varphi_5 + \pi) \\ T_5 = T_2 = L_5 F_4 \cos(\varphi_4 - \varphi_5 - 90), & \varphi_4 \geq (\varphi_5 + \pi) \end{cases} \tag{18}$$

3.7 Kinematic Analysis

The kinematic analysis of the selected leg mechanism aims to derive the changes in position, velocity and acceleration, without considering the applied force [1]. These forms of analyses are primarily made up of two components: inverse kinematic equations, which calculate the joint angles of a mechanism given the position of the end effector, and forward kinematic equations, which enable the calculation of the foot position from predefined joint positions [2]. These equations are critical to maintain balance, follow precise trajectories and achieve effective motion planning, whilst also highlighting the relationship between components that make up the mechanism [2].

3.7.1 Inverse Kinematics

The inverse kinematics of this system can be solved using expressions derived from [28], the same paper in which the geometries of the leg mechanism were derived. These formulae calculate the actuator joint angles from a given cartesian foot position and are broken down into a set of coefficient equations as well as a quadradic equation for the angle of the actuators. The set of coefficient equations are described by the formulae below [28].

$$\begin{cases} A_2 = -l_2(2x_m + l_1) \\ B_2 = -2l_2 y_m \\ C_2 = {x_m}^2 + {y_m}^2 + \frac{{l_1}^2}{4} + {l_2}^2 - {l_3}^2 + l_1 x_m \end{cases} \tag{19}$$

$$\begin{cases} A_5 = -l_5(2x_m + l_1) \\ B_5 = -2l_5 y_m \\ C_5 = {x_m}^2 + {y_m}^2 + \frac{{l_1}^2}{4} + {l_5}^2 - {l_4}^2 - l_1 x_m \end{cases} \tag{20}$$

Using the results of the coefficients above, the joint angles for the actuators can be calculated by applying the quadradic equation [28].

$$\varphi_{2,5} = 2\ \tan^{-1}\left(\frac{B_{2,5} \pm \sqrt{{A_{2,5}}^2 + {B_{2,5}}^2 - {C_{2,5}}^2}}{A_{2,5} - C_{2,5}}\right) \tag{21}$$

3.7.2 Forward Kinematics

The forward kinematics of this system were solved using the vector-loop method discussed in the literature review and outlined in [29]. This method defines each link as a vector represented with a magnitude and direction, where the head to tail addition of each vector (linkage) results in a closed loop [29]. For the 5-RRRRR mechanism used in this project, the resulting vector addition leads to two unknown angles, which can be determined by applying case 2c outlined in [29]. Figure 50 below depicts the application of this method to derive the position, velocity and acceleration relationships with the drive angles of the actuators by hand.

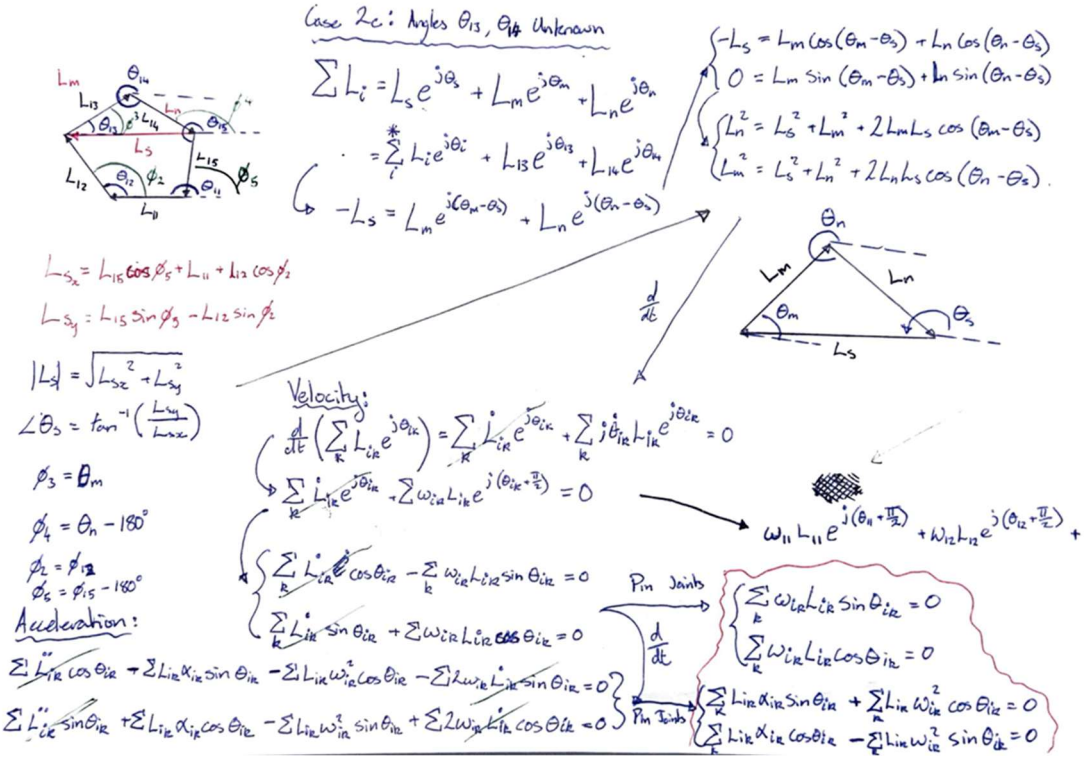

Figure 50: Forward Kinematic Analysis

The resulting position equations are as follows [29]:

$$l_s = \sqrt{(l_1 - l_5\cos(\varphi_5 + \pi) - l_2\cos(\varphi_2))^2 + (l_5\sin(\varphi_5 + \pi) + l_2\sin(\varphi_2))^2} \quad (22)$$

$$\varphi_s = \tan^{-1}\left(\frac{(l_1 - l_5\cos(\varphi_5+\pi) - l_2\cos(\varphi_2))}{(l_5\sin(\varphi_5+\pi) + l_2\sin(\varphi_2))}\right) \quad (23)$$

$$\varphi_3 = \varphi_s - \cos^{-1}\left(\frac{{l_4}^2 - {l_s}^2 - {l_3}^2}{2l_s l_3}\right) \quad (24)$$

$$\varphi_4 = \varphi_s + \cos^{-1}\left(\frac{{l_3}^2 - {l_s}^2 - {l_4}^2}{2l_s l_4}\right) + \pi \quad (25)$$

The resulting velocity equations are as follows [29]:

$$\omega_2 l_2 \cos(\varphi_2) + \omega_3 l_3 \cos(\varphi_3) + \omega_4 l_4 \cos(\varphi_4 + \pi) + \omega_5 l_5 \cos(\varphi_5 + \pi) = 0 \quad (26)$$

$$\omega_2 l_2 \sin(\varphi_2) + \omega_3 l_3 \sin(\varphi_3) + \omega_4 l_4 \sin(\varphi_4 + \pi) + \omega_5 l_5 \sin(\varphi_5 + \pi) = 0 \quad (27)$$

The resulting acceleration equations are as follows [29]:

$$\alpha_2 l_2 \sin(\varphi_2) + \alpha_3 l_3 \sin(\varphi_3) + \alpha_4 l_4 \sin(\varphi_4 + \pi) + \alpha_5 l_5 \sin(\varphi_5 + \pi) + {\omega_2}^2 l_2 \cos(\varphi_2) + {\omega_3}^2 l_3 \cos(\varphi_3) + {\omega_4}^2 l_4 \cos(\varphi_4 + \pi) + {\omega_5}^2 l_5 \cos(\varphi_5 + \pi) = 0 \quad (28)$$

$$\alpha_2 l_2 \cos(\varphi_2) + \alpha_3 l_3 \cos(\varphi_3) + \alpha_4 l_4 \cos(\varphi_4 + \pi) + \alpha_5 l_5 \cos(\varphi_5 + \pi) - {\omega_2}^2 l_2 \sin(\varphi_2) - {\omega_3}^2 l_3 \sin(\varphi_3) - {\omega_4}^2 l_4 \sin(\varphi_4 + \pi) - {\omega_5}^2 l_5 \sin(\varphi_5 + \pi) = 0 \quad (29)$$

Once derived, the equations allow for the position, velocity and acceleration of each linkage to be calculated provided the position, velocity and acceleration of the actuators are known.

3.7.3 Position, Velocity and Acceleration Analysis

To visualise the kinematic relationship between the linkages that make up the selected leg mechanism, the forward kinematic equations can be applied.

Initially, a grid of discrete data points was created for the possible actuator angles. Next the positional equations 24 and 25 can be implemented to calculate the angles of the passive joints for each configuration. The resulting points are plotted as a surface that characterises the workspace for each angle. These plots are depicted below in Figure 51.

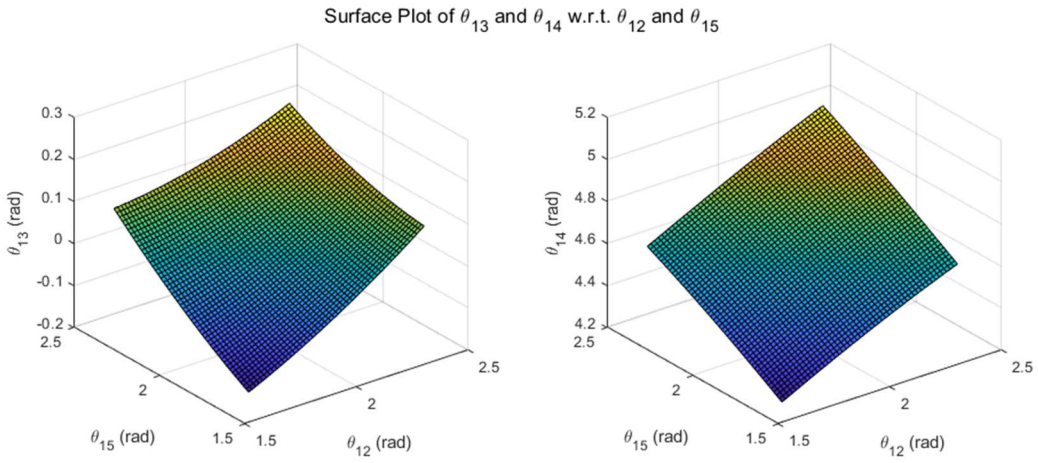

Figure 51: Position Surfaces of Lower Link Angles.

Following the positional analysis, the velocity of the passive joints can be calculated from an arbitrary position with a velocity equal to the maximum speed of the actuator outlined in [60]. The resulting surface demonstrates the transmission characteristics or relationship between the actuator speed and the resulting speed of the passive links. These plots are depicted below in Figure 52.

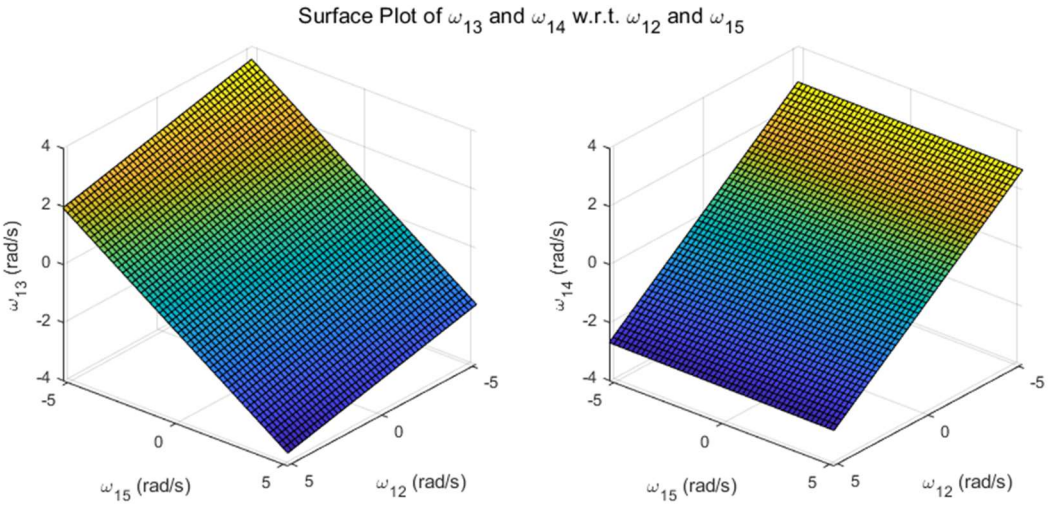

Figure 52: Velocity Surfaces of Lower Link Angles.

Finally, the acceleration of the passive joints can be calculated from a velocity equal to the maximum speed of the actuator outlined in [60], and an arbitrary acceleration of 1 rad/s^2. The resulting surface demonstrates the transmission characteristics or relationship between the actuator acceleration and the resulting speed of the passive links. As shown by Figure 53 below, these surfaces follow a similar relationship to the transmission of velocities to the passive linkages.

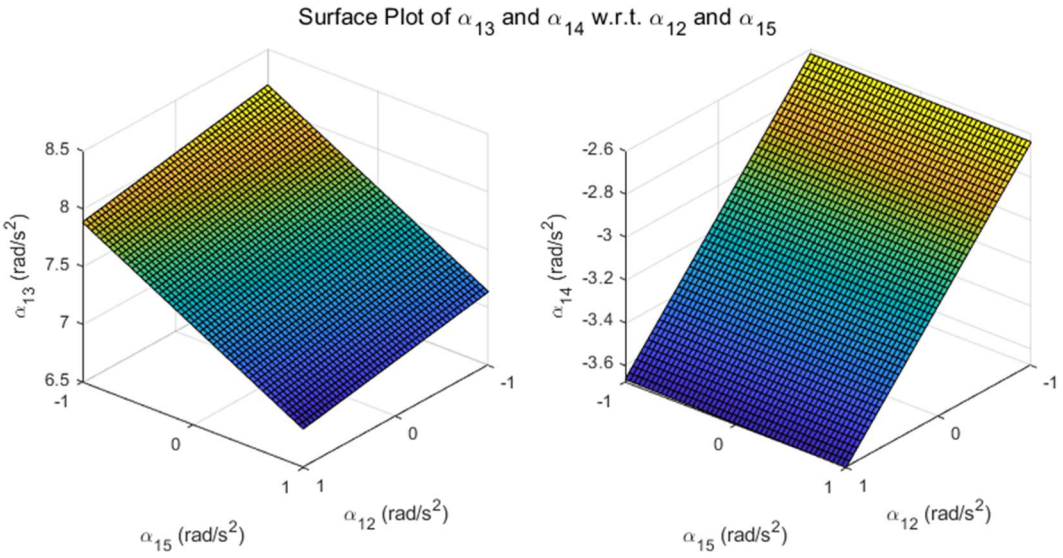

Figure 53: Acceleration Surfaces of Lower Link Angles.

The MATLAB used to complete this analysis is available in Appendix C.

3.8 Dynamic Analysis

The dynamic analysis of a manipulator is utilised to determine the motion of the mechanism that arises from the applied external forces and torques [61]. To complete the dynamic analysis selected leg mechanism, three stages were applied. The first is a static approximation of dynamic behaviour, which breaks dynamic motion into discrete intervals, treating each interval as a static position. The next stage is to develop a simple straight path simulation of the robot utilising SolidWorks Motion Analysis to derive the reactive torques and forces experienced on the leg mechanism. Finally, formulae can be applied to solve for the Jacobian matrix of the manipulator to relate angular joint motion to cartesian motion of the foot.

3.8.1 Static Approximation of Dynamic Behaviour

To begin a dynamic analysis, a static approximation of the dynamic behaviour of a mechanism is often employed to provide an estimate of the behaviour whilst avoiding the complexity of dynamic simulations.

The dynamic analysis of the selected leg mechanism over its gait trajectory can be approximated by conducting a static analysis at each discrete interval across this path. To do this, the scenario and equations derived in the static analysis section 3.6.2 can be applied with a high interval resolution, under the assumption that the time taken for the mechanism to transition between intervals is equal. Applying a total time of 2.5s for a single step cycle, a total resolution of 400 discrete points, and an applied contact force as described in 3.6.2, a static approximation of the external forces and torques acting upon the actuators can be calculated. The result of the estimated dynamic behaviour is depicted in Figure 54 below.

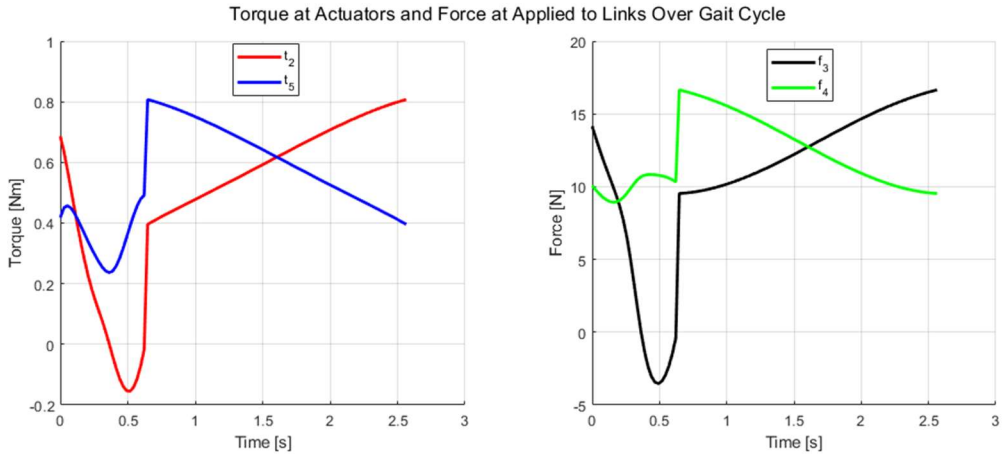

Figure 54: Static Approximation of Forces and Torques Over a Single Gait.

This estimation also provides a static approximation of the kinematic behaviour of the selected leg mechanism over a single step cycle. The displacement, velocity and acceleration from this approximation is depicted in Figure 55, Figure 56 and Figure 57.

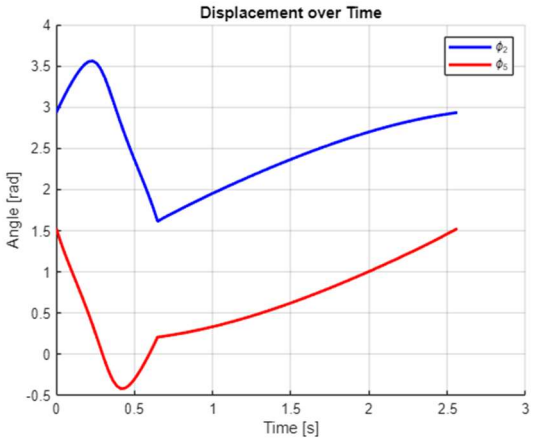

Figure 55: Static Approximation of Angular Motor Displacement Over a Single Gait.

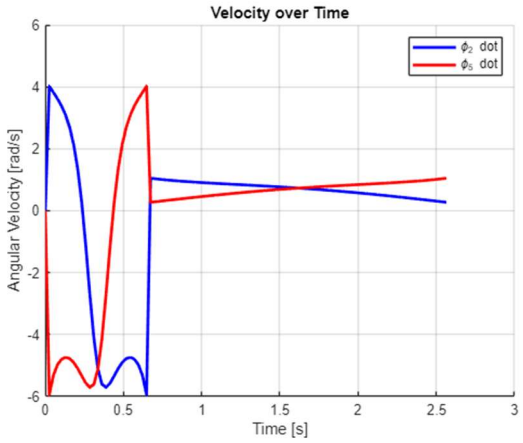

Figure 56: Static Approximation of the Angular Motor Velocity Over a Single Gait.

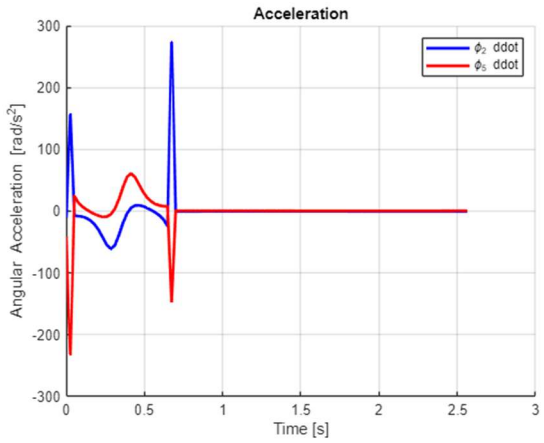

Figure 57: Static Approximation of the Angular Motor Acceleration Over a Single Gait.

The MATLAB used to complete this analysis is available in Appendix D.

3.8.2 SolidWorks Motion Analysis

The SolidWorks Motion Analysis add-in was employed for the simulation of a trotting motion of the quadruped robot in its square configuration. This simulation enables the dynamic analysis of the forces and torques experienced on the actuators during the walking motion of the robot.

To begin the simulation, the same simplified configuration utilised in the static analysis was adopted. To define the sinusoidal gait motion for each leg mechanism, a

sketch of the gait pattern was created and a path mate between the sketch and each foot was applied. The feet were then offset into the starting position of a trot gait before a gravitational and loading force was applied to the body. A fixed floor was established out of a dense titanium material and contact interactions were defined between the feet and this surface.

The results of the simulation depicting the forces and torques at the actuators and passive joints are given by Figure 58, Figure 59, Figure 60 and Figure 61 below.

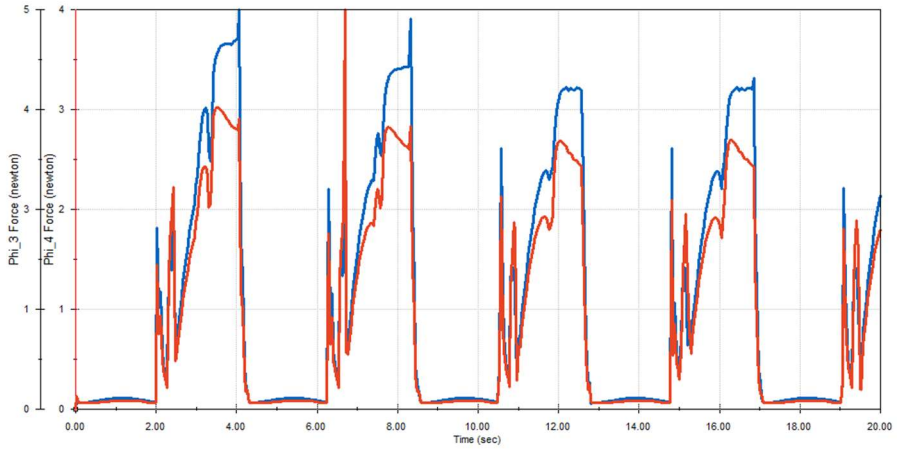

Figure 58: Forces Experienced by the Passive Joints During SolidWorks Motion Analysis.

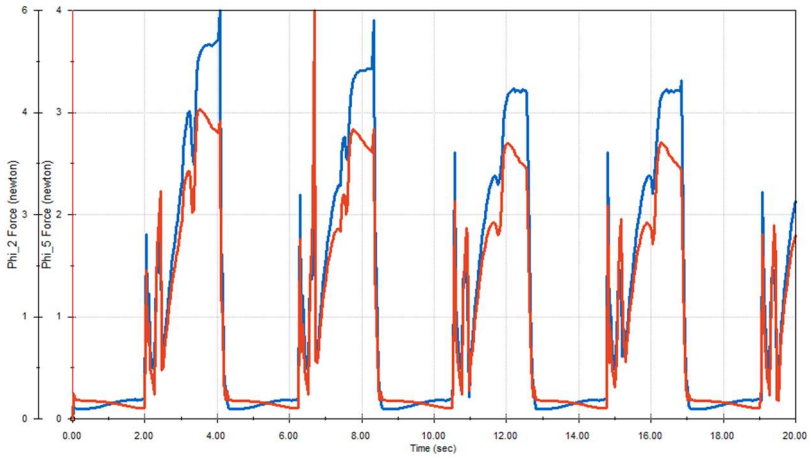

Figure 59: Forces Experienced by the Actuators During SolidWorks Motion Analysis.

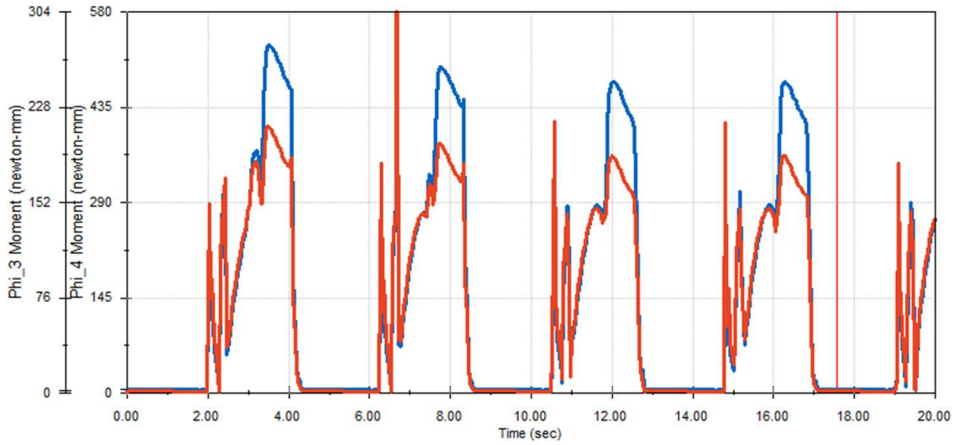

Figure 60: Torques Experienced by the Passive Joints During SolidWorks Motion Analysis.

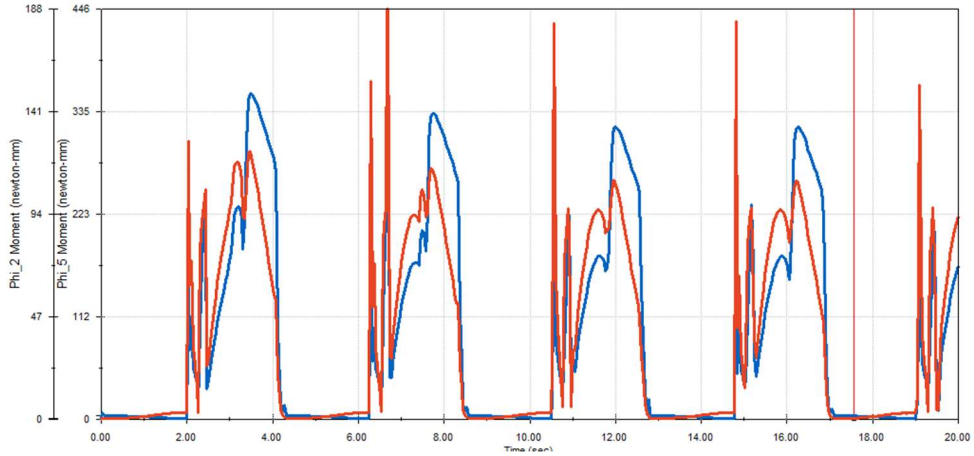

Figure 61: Torques Experienced by the Actuators During SolidWorks Motion Analysis.

3.8.3 Determination of the Jacobian Matrix

Jacobian matrices are multidimensional derivatives that relate the joint velocities of a mechanism to the cartesian velocities of an end effector [61]. The Jacobian matrix for the selected mechanism can be calculated by applying the coefficients described in section 3.7.1 and leveraging the equations outlined in [62]. The resulting equations for the Jacobian matrix of this project have been sourced from [62] and are given below:

$$J = J_x^{-1} \cdot J_q \tag{30}$$

$$J_q = \begin{bmatrix} \frac{\delta F_2(X,Q)}{\delta \varphi_2} & \frac{\delta F_2(X,Q)}{\delta \varphi_5} \\ \frac{\delta F_5(X,Q)}{\delta \varphi_2} & \frac{\delta F_5(X,Q)}{\delta \varphi_5} \end{bmatrix} \quad (31)$$

$$J_x = \begin{bmatrix} \frac{\delta F_2(X,Q)}{\delta x_m} & \frac{\delta F_2(X,Q)}{\delta y_m} \\ \frac{\delta F_5(X,Q)}{\delta x_m} & \frac{\delta F_5(X,Q)}{\delta y_m} \end{bmatrix} \quad (32)$$

$$F_{2(x_m,y_m)} = A_2(x_m, y_m)\cos(\varphi_2) + B_2(x_m, y_m)\sin(\varphi_2) + C_2(x_m, y_m) \quad (33)$$

$$F_{5(x_m,y_m)} = A_5(x_m, y_m)\cos(\varphi_5) + B_5(x_m, y_m)\sin(\varphi_5) + C_5(x_m, y_m) \quad (34)$$

3.9 Design of the Electronics Subsystem

The design of the electronics subsystem to control the quadruped robot stems largely from the implementation of past papers.

Upon completing a literature review it was identified that the minimum requirements for a quadruped robot's electronics subsystem included: a microcontroller, a set of actuators, a driver for the actuators, a power source and a power regulator. Additionally, components such as an IMU and battery monitor were included to provide perception regarding the robot's environment. Other components such as cameras or encoders could also be added, however, to simplify the control system they were excluded due to their complexities.

3.9.1 Microcontroller

The microcontroller utilised in this project is an ESP32 DevKitC V4 module with the WROOM-IE chip. This board provides both onboard Bluetooth and WIFI connectivity, including an external antenna to extend its range, enabling a reliable connection to peripheral devises without the need for an additional module [33]. The

microcontroller features a 32-bit LX6 dual core processor that runs up to 240MHz, with 520 KB of SRAM and 8MB of Flash memory, providing ample computational power for this application [33, 34]. The board supports a total of 34 input and output pins, offering sufficient I/O capacity to interact with other subsystem components [30]. Additionally, the ESP32 breakout board is in expensive and energy-efficient compared to other available microcontroller platforms [30]. An image of this microcontroller is provided in Figure 62 below.

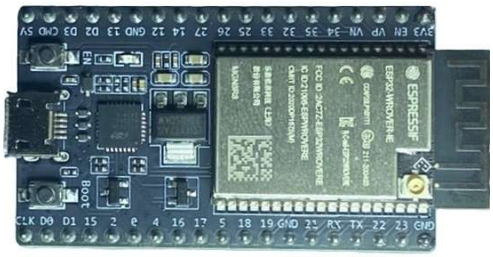

Figure 62: ESP32 DevKitC V4.

3.9.2 Actuators

This project makes use of FS5109M servo motors to provide actuation to the leg mechanisms. A servo type actuator was selected as it provides high gear ratios for increased motor torques, it exhibits fast responses to change in control signals and it provides a level of positional accuracy that meets the need [44]. However, these motors do not provide external positional feedback and are therefore limited to open loop control [44].

To effectively determine the required torque rating for the actuators, the results from the dynamic analysis were reviewed. A modest safety factor of 1.2 can be applied to the maximum torque experienced by the actuators during this analysis, in order to achieve the greatest performance for the lowest cost [63]. In this instance, the peak torque experienced by the actuators was identified in the static approximation of the dynamic analysis, reaching a value of approximately 0.8 Nm. Applying a safety factor of 20%, the required size of the servo motors is 0.96 Nm or approximately 10 kgcm, which is the rating of the FS5109M motor. To verify the selection, the motor torque

was also compared to the peak torque experienced in the SolidWorks Motion Analysis of 0.58 Nm, resulting in a safety factor greater than 1.4.

Another major factor in the selection of the FS5109M servo was its use of metal gears for the servo gearbox. Compared to plastic gears often used in hobbyist grade motors, metal gears reduce the risk of tooth stripping under high torques, which subsequently limits the likelihood of failure and the frequency of maintenance.

In total eight actuators were utilised in this project, with two allocated per leg. To drive these motors, Adafruit's PCA9685 PWM breakout board is used. This PWM driver board can control up to 16 servos at a maximum of 12 volts, with 12-bit resolution allowing for precise control of the motors at their rated voltage [64]. This driver is commonly found in literature with implementations in [30], [33] and [65] depicting its success for the control of multi-legged robots.

3.9.3 Power System

The power system in this project is designed to complete two primary functionalities: providing a regulated voltage supply for each electronic component and meeting the power requirements of the selected actuators. The electronic components, including the microcontroller, driver board, and sensors all operate at a nominal voltage of 5V. On the other hand, the servo motors require a minimum operation voltage of 5V, with the desired torque performance reached above 6V.

To meet these requirements, a 2S 7.4V 5000mAh lithium-ion battery, similar to the one utilised in [31] was selected. The capacity of this battery was determined by applying a worst-case scenario in which each actuator operates at its stall torque of 1.7A [60]. For this scenario, an operating duration of 20 minutes was decided upon, to enable robot recovery, but reduce the likelihood of damage under high loads for long periods of time. Given this project implements eight actuators with a total current draw of 13.6A, the required battery capacity is approximately 4500mAh. A 2S configuration was also selected to achieve a nominal voltage above the 6V required by the actuators.

To regulate the power supply for the electronics, a step-down buck converter was utilised. Specifically, Adafruit's MPM3610 5V converter which accepts up to 21V of DC power and produces a stable 5V output at 1.2A [66]. This DC-DC voltage regulator

provides ample power for the microcontroller and sensors, with proof of its implementation found in [67].

To ensure safety and improve efficiency, a single pole single throw switch is utilised to isolate the battery and power on or off the system as required.

3.9.4 Sensors and Peripherals

The electronic subsystem consists of two sensors that allow the robot to perceive the surrounding environment.

Since the servo motors offer no positional feedback, the state of the robot can be estimated by utilising an onboard Inertial Measurement Unit (IMU) [31, 44]. The IMU selected for this project is Adafruit's ICM-20948, which provides 9 DOF via a gyroscope, accelerometer and compass [68]. Using the role and pitch angles from this sensor can enable the robot to perform posture stabilisation [45].

Along with the IMU, this project implements a battery monitor to provide insights into the power usage of the robot. For this component Adafruit's INA260 is utilised to avoid overdischarge of the battery, whilst enabling monitoring of the battery voltage and system power draw [42, 43].

To control the robot, a peripheral Bluetooth Xbox controller is utilised. This form of controller is similar to the PlayStation controller implemented by [1] and allows for successful motion control of the quadruped without the need for a tethered physical connection.

3.9.5 Complete Electronics System

The electronics system is housed within the electronics module, with each breakout board secured to the base plate via custom holders and nylon bolts. To form the electrical connections between modules and the custom connectors, simple jumper wires were used. Figure 63 below depicts the internals of the electronics module including the holders, whilst Figure 64 provides a schematic diagram of the electrical connections.

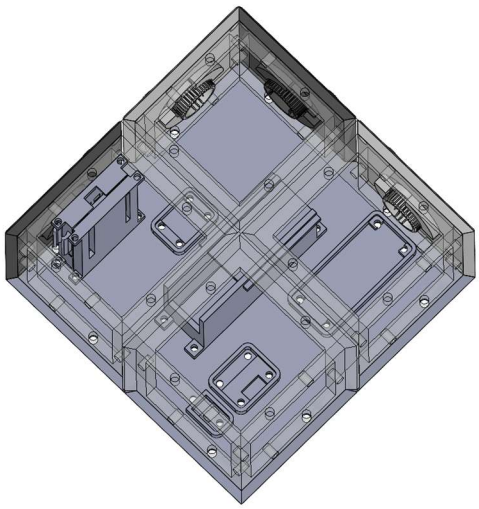

Figure 63: Internals of the Electronics Module.

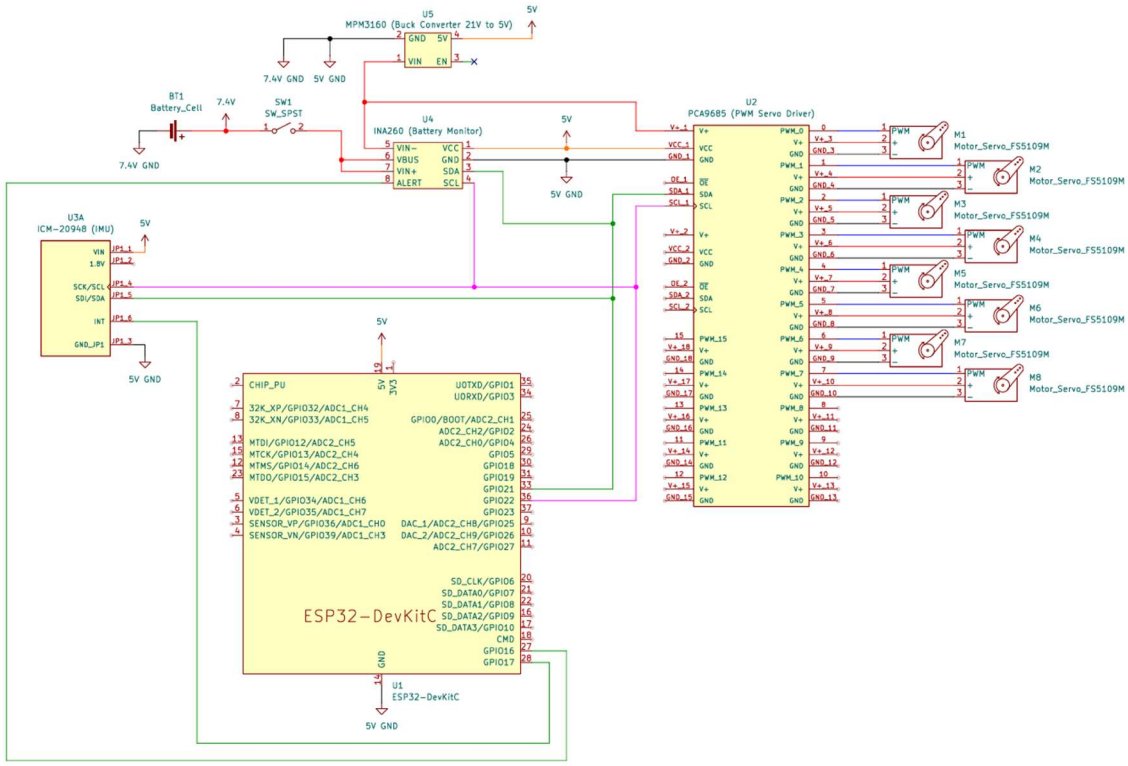

Figure 64: Schematic Diagram of the Electronics Subsystem.

3.10 Control System

To meet the scope and functional requirements of this project, the control system utilised by the quadruped robot was simplified, ensuring reliable control and stability without introducing a large computational overhead. This system combines an open-loop controller, and a state machine, to provide simplistic motion and transition between robot behaviours.

The control strategy utilised in this project, follows an approach similar to that presented in [1], where each leg mechanism follows a predefined gait trajectory with the starting point of this path offset to achieve the desired walking gait pattern. Since the actuators do not provide positional feedback, the control system does not adjust to correct errors between the actual foot position and its desired position, such is the nature of open-loop control [44].

To enable minor variations in the walking pattern, such as changes in step height, body elevation and movement direction, the controller varies gait parameters, including step height, step offset and step length. These changes are actioned by the user through a remote and provide steering via the differential drive of the leg mechanisms.

3.10.1 Control System Process Overview

The control sequence begins with the initialisation of the ESP32's onboard Bluetooth system, before I2C communication between the servo driver, IMU and battery monitor is established. After initialisation, the program enters the main control loop which completes three key steps. The first step is to obtain data from the remote control, IMU and battery monitor, provided they have been initialised correctly. Next the state of the robot is updated based on the previous or initial state and the controller inputs. If multiple controller inputs are registered the state machine gives priority to states based on precedence. This precedence and the controller input that triggers each state is depicted in Table 6 below.

Table 6: Map of Controller Inputs and Control System States.

Controller (Keyboard) Button	State	Precedence
Home (H)	Home Sequence	0
Up D-Pad (A or ←)	Forward	1
Down D-Pad (A or ←)	Reverse	2
Left D-Pad (A or ←)	Turn Left	3
Right D-Pad (A or ←)	Turn Right	4
Right Trigger (Q)	Raise Body	5
Left Trigger (E)	Lower Body	6
Right Bumper (Z)	Raise Step Height	7
Left Bumper (X)	Lower Step Height	8
Share (R)	Reboot	9
No Input Triggered	Stationary	10

Following the update of the state, the corresponding robot behaviour is executed. For states zero to eight, this behaviour includes the setting of key gait parameters before the execution of the trajectory generator, inverse kinematics solver and servo driver.

During the home state, the current discrete increment of the gait trajectory is reverted back to its initial starting position. In both the left and right states, asymmetrical step lengths, $x_f - x_s$, are introduced between either side of the robot, with each of their respective sides shorter than the other, allowing the robot to turn through differential motion. The forward and reverse states on the other hand, maintain an equal step length on either side of the robot, with the forward state increasing the discrete trajectory increment to promote forward linear motion whilst the reverse state results in a decrease in this increment and a subsequent backward motion.

To raise and lower the robot's body, these states increase or decrease the offset, y_s, between the body and the top of the gait trajectory. A similar approach is taken to change the step height, h, with each state increasing or decreasing the distance that the foot is lifted in the air.

All variations in the step length, offset and height defined by each of these eight states are bounded in such a way that the resultant gait trajectory never exceeds the maximum dexterous workspace defined in previous sections.

Once parameters are set, a call to the trajectory generator is made to calculate the cartesian coordinates of the next discrete gait increment. The cartesian position is then passed to the inverse kinematic function to determine the corresponding actuator

angles required to reach this position. These angles are then converted into a PWM signal which is sent to the actuators by the servo driver.

The remaining two states follow a slightly different process, with the reboot state triggering a restart of the ESP32 and entire control sequence, while the stationary state maintains the robot's previous configuration.

Upon the completion of each of these behaviours, the control loop repeats itself after a minimal 10ms delay. A visual representation of this control structure is provided in Figure 65 below

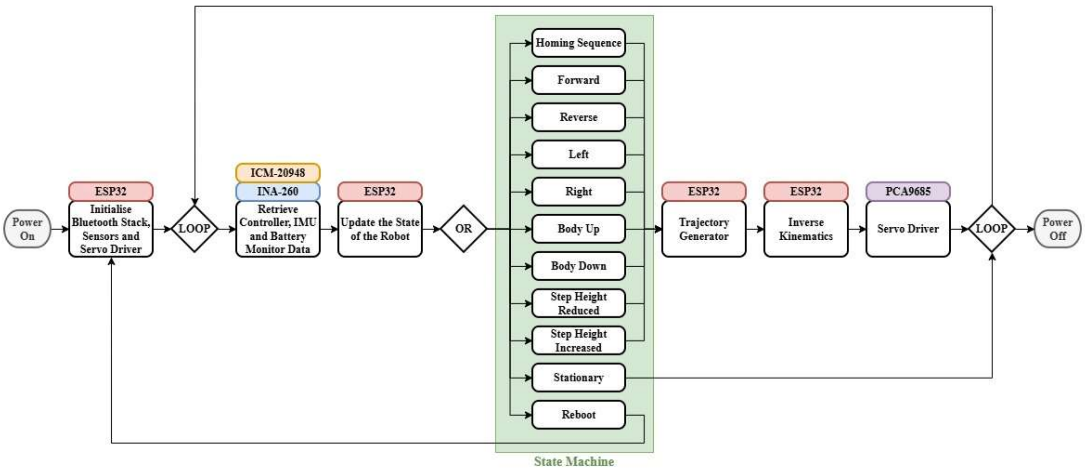

Figure 65: Open-Loop Control System of the Robot Quadruped.

3.10.2 Control System Implementation

The control system for this project is implemented in C++ using the ESP IDF framework, visual studio code IDE and PlatformIO design environment. This form of implementation was selected due to its ability to integrate third-party packages that are utilised in this project.

One of the critical libraries leveraged in this project is an open-source third-party package known as Bluepad32 which enables a Bluetooth connection to be established with a wide range of peripherals [69]. This package is designed specifically to run on ESP32 systems and provides a template PlatformIO project to enable rapid incorporation into projects [69, 70].

The communication with electronic components such as Adafruit's IMU, battery monitor and PWM driver was also achieved using existing open-source packages provided by Adafruit on GitHub [71-73]. These packages allowed for reliable interfacing with the company's breakout boards.

The remaining components of the control system such as the trajectory generator and inverse kinematics solver were implemented using the equations outlined in previous sections.

The complete source code for this project is available in Appendix E.

Chapter 4. Results and Discussion

4.1 Manufacture and Assembly of a Prototype

Following the theoretical design and analysis of a quadruped robot, a physical prototype was manufactured and assembled. The predominant structure of the robot was 3D-printed from PLA and joined using heated inserts and fasteners.

The electronics subsystem was assembled inside its respective module, with electrical connections made via jumper wires arranged according to the schematic diagram. Nylon bolts were utilised to secure the breakout boards within their customised holders and helped reduce the chance of electrical shorting.

Each of the four actuator modules were assembled by mounting the servo motors to their corresponding face plates and fixing their output shafts to the leg mechanism via metallic servo horns. The leg mechanisms themselves were assembled with bearings and bolts to secure them in place whilst allowing for smooth movement.

In total the quadruped robot implemented two spacer modules, four actuator modules and an electronics module arranged in the square configuration. The following figures depict the individual modules and complete assembly of the prototype.

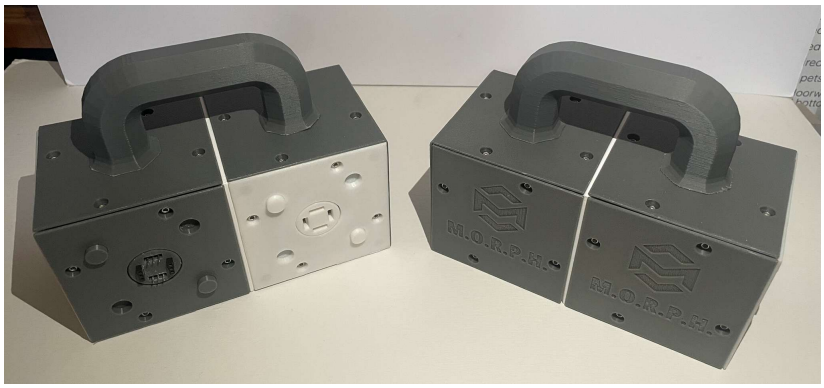

Figure 66: Prototype Implementation of the Spacer Modules.

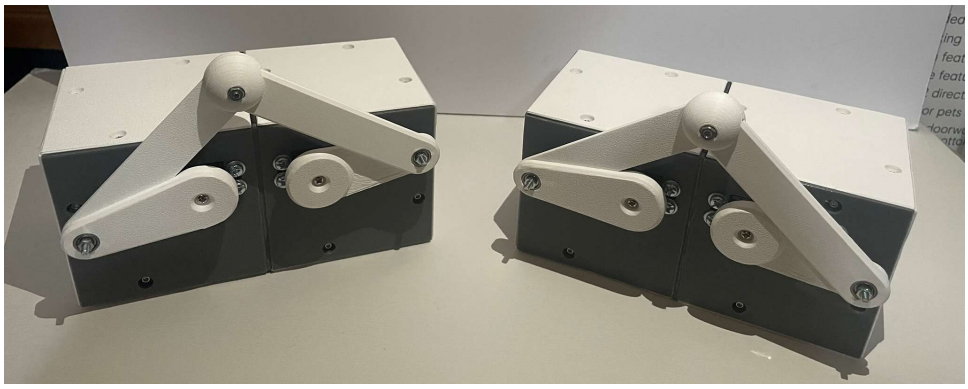

Figure 67: Prototype Implementation of the Actuator Modules.

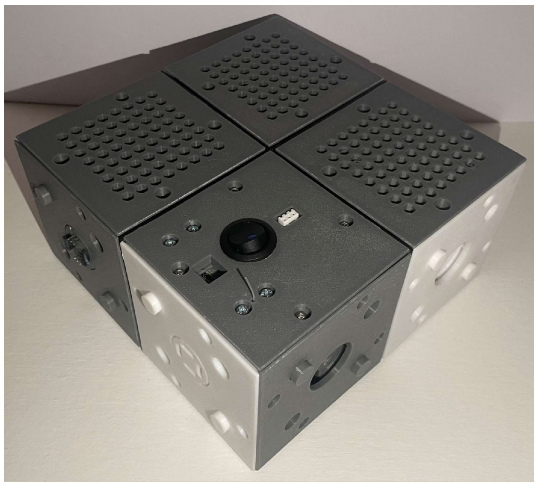

Figure 68: Prototype Implementation of the Electronics Module.

Figure 69: Prototype Implementation of the Assembled Quadruped Robot.

The successful manufacture and assembly of the prototype quadruped robot verified the geometric design and fit of individual components. For the most part, tolerances between components allowed for straight forward assembly.

Deviations from the expected performance were observed in the connection strength between modules in the complete assembly. Whilst individual and partial connections demonstrated acceptable strength and rigidity, the accumulative weight of the entire robot reduced the reliability of module connections.

Although the modules remained joined during the static configuration of modules, the likelihood of failure under dynamic impacts and vibrations was higher than intended. As such, supporting straps were employed to prevent module disconnection throughout the subsequent testing procedures.

4.2 Power and Control System Testing

Once assembled, the robot was powered on to verify the functionality of the electronics and control system. After receiving power and establishing a connection to a controller via Bluetooth, each input command was systematically cycled through to confirm the correct response was taken by the control system. At the conclusion of this test, each state of the system was correctly activated, with the robot performing the desired movement associated with each state, including the prioritisation of the states, such as the home state.

During this test, the outputs of the IMU, battery monitor and Bluetooth controller via the serial terminal were observed. Each subsystem responded as expected with each sensor providing meaningful data about the robot's current state.

4.3 Dynamic Motion Testing

Following the construction of the prototype, a variety of tests were conducted to verify the quadruped robot's dynamic performance. The first of these tests was to evaluate the straight line and turning motion of the quadruped on the uneven surfaces outlined in the scope of this project.

To enable quantitative comparisons between each surface, the quadrupedal robot was made to complete the following timed tasks.

- Traverse a distance of a metre in a forward and reverse straight-line motion.
- Complete a 90º turn in both the left and right directions.

The results of this test are provided in Table 7 below:

Table 7: Results of Dynamic Motion Tasks.

	Forward Straight-Line Motion Over 1m				**Reverse Straight-Line Motion Over 1m**				**90º Turn**	
Surface	**1**	**2**	**3**	**Ave**	**1**	**2**	**3**	**Ave**	**Left**	**Right**
Floorboards	21.04	21.49	21.20	21.24	21.57	21.15	21.29	21.34	Yes	Yes
Grass	62.45	55.30	49.83	55.86	52.47	55.83	54.73	54.34	Yes	Yes
Carpet	21.94	21.77	23.03	22.25	22.18	21.86	21.91	21.98	Yes	Yes
Pavement	38.41	50.35	43.79	44.18	35.69	42.15	39.64	39.16	Yes	Yes

A depiction of the robot on each surface is provided in Figure 70, Figure 71, Figure 72 and Figure 73 below.

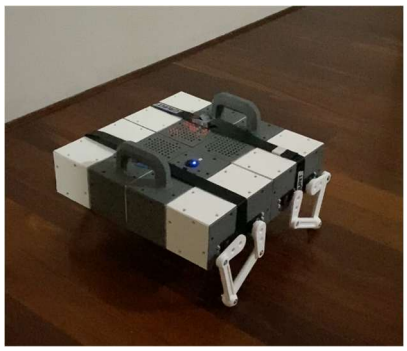

Figure 70: Dynamic Motion Test on Floorboards.

Figure 71: Dynamic Motion Test on Grass.

Figure 72: Dynamic Motion Test on Carpet.

Figure 73: Dynamic Motion Test on Pavement.

For each surface, the quadruped robot demonstrated the ability to move in forward, reverse, left and right directions with varying degrees of success. The quadruped's motion was best exhibited on the wooden floorboards and carpet, reaching a maximum velocity of 0.047 m/s and 0.045 m/s respectively. On these surfaces the robot achieved stable and consistent locomotion, which is reflected in the results.

The performance of the robot degraded significantly on uneven surfaces such as grass and pavement surfaces, completing each task with large variations in time. These inconsistencies are caused predominantly by the feet of the robot becoming caught in hollows or gaps in the surfaces, taking extended periods of time to regain directional motion. This behaviour is likely due to the open-loop control system that prevents the robot from adapting its gait to suit variations in surfaces.

This test also aided in the validation of the gait sequency, foot trajectory and inverse kinematic equations. During locomotion, each leg mechanism appeared to follow a sinusoidal trajectory, with the phase offsets of each leg equivalent to a walking gait. The bounds of the inverse kinematics and the lengths of each leg mechanism also prevented the mechanism from reaching any singularity positions in which it couldn't recover.

4.4 Carrying Capacity

After dynamic motion testing of the robot, a carrying capacity test was conducted to validate the results of the static and dynamic analysis. In both these analyses, it was deemed theoretically possible for the robot to carry a load of up to one body weight.

Given the total weight of the prototype measured 3.5 kg, the theoretical carrying capacity was tested by applying a load consisting of a 3kg weight and a 0.5kg book to the robot. The load was then, lifted and lowered repeatedly to observe the behaviour of the actuators. On the completion of the test, no unusual behaviours were identified, verifying the carrying capacity of up to 100% of the robot's body weight. To prevent any damage to the prototype, the load was not increased further. By validating the theoretical carrying capacity, the design helps conform to its goals to allow the attachment of tooling, which adds additional load.

4.5 Power Consumption

Finally, the power consumption of the robot whilst performing a variety of movements was tested. To do this, the robot was placed upon a stand and moved in the forward, reverse, left and right direction over a number of gait cycles. During this test the average time taken to complete one gait cycle was approximately 2.5s.

Using the onboard battery monitor, the voltage current and power draw of the system could be monitored over time. A summary of the average power characteristics for each motion is provided in Table 8 below.

Table 8: Average Power, Current and Voltage for Each Motion.

Motion	Average Power	Average Current	Average Voltage
Forward	6.14 W	0.87 A	7.31 V
Reverse	5.90 W	0.82 A	7.42 V
Left	5.17 W	0.75 A	7.35 V
Right	4.98 W	0.73 A	7.07 V

To visualise these characteristics, the power, voltage and current values for the forward motion have been graphed against time. These plots are given by Figure 74, Figure 75 and Figure 76 below, with the graphs for the remaining motions available in Appendix F.

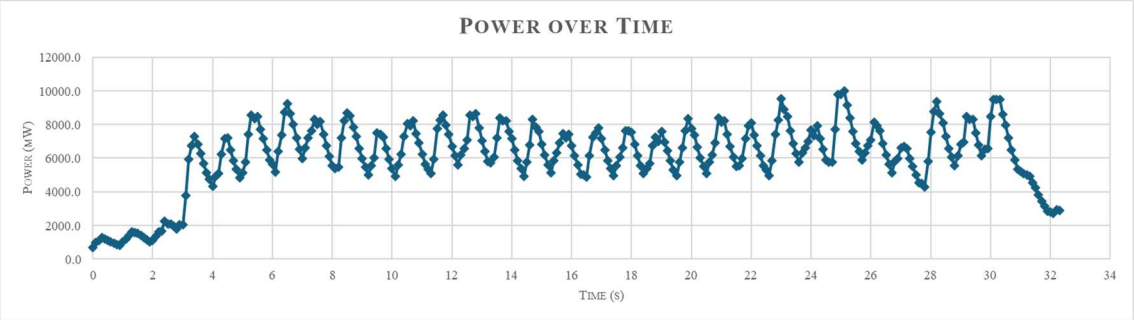

Figure 74: Power Readings During a Forward Motion Over Time.

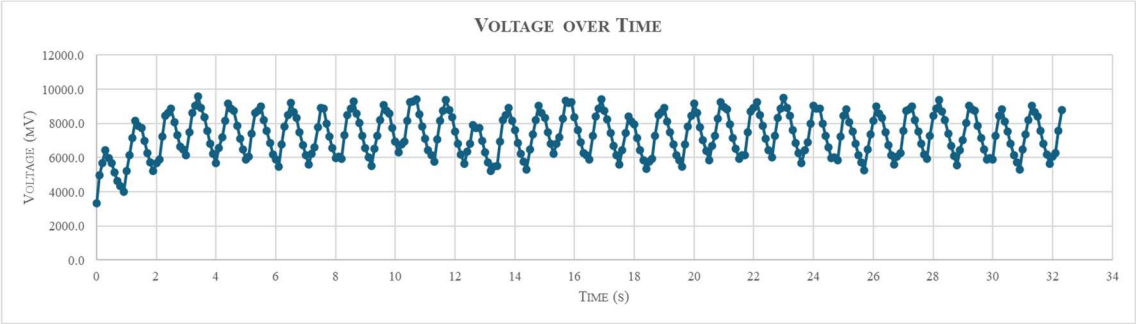

Figure 75: Voltage Readings During a Forward Motion Over Time.

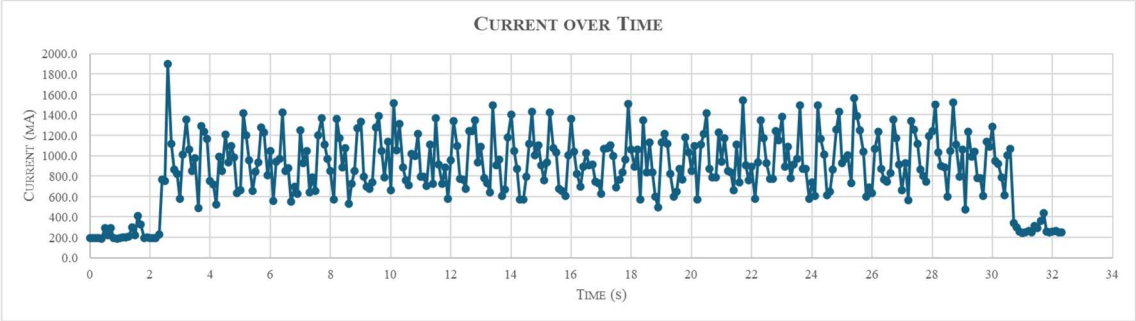

Figure 76: Current Readings During a Forward Motion Over Time.

As shown by the graphs above, each characteristic experience a periodic rise and fall approximately every 1s. These spikes likely correspond to the sharp change in direction experienced between the curved swing phase and horizontal support phase, with the troughs occurring during the midpoint of these phases. With large changes in displacement, the actuators appear to draw more current as they conduct a rapid acceleration change.

Given the battery used in the electronic subsystem has a capacity of 37Wh and the average power consumption of a forward motion, it is estimated that the operation time for this prototype is approximately 6 hours.

Chapter 5. Conclusions

At the conclusion of this project, a number of design approaches and components have been achieved. The results of these processes include all of the desired deliverables: including 3D models and assemblies of the entire quadruped robot; complete static, kinematic and dynamic analyses with equations and results; a schematic of the electronic components utilised; source code for an open-loop control system; and a physical prototype of the robot. This project extends the deliverables to also incorporate MATLAB scripts for parameter derivation of five-bar linkages, visual simulations of the robot in SolidWorks, and the design pattern for a custom modular connector.

To evaluate the resultant design against the scope of the project, the test results in section 4 can be compared with the outlined performance indicators. The implemented prototype achieves both the stability indicators with a high degree of success, with the ability to stand at any position in its given gait without aid, and the completion of numerous repetitive gait cycles in multiple directions.

A significant implementation of modularity throughout the entirety of the quadruped robot's design is also achieved. Its standardised component modularity allows the same custom connector to be utilised on either ends of a connection, and its volumetric unit geometry prevents issues with reconfiguration due to its uniformity. The segregation of operational components into separate modules allows the robot to achieve various configurations, including a wide base arrangement for carrying loads and a narrow footprint for inspections. The prototype's electrical performance also exceeds the minimum required operating time of 30 minutes by more than ten times, with a forecast battery life of 6 hours.

The performance of the implemented system does meet the movement requirements of traversing a distance of 1 metre and completing a full turn; however, these are completed with various degrees of success. For flat, uniform surfaces, the prototype exceeds, achieving a maximum velocity of 0.047 m/s, whilst maintaining highly stable movements. Relatively uneven surfaces, such as grass and pavement does pose a challenge to the robot, particularly when traversing gaps. Although the robot can complete the set tasks, it does so with a large degree of variation that could be improved upon.

It should be noted that the carrying capacity and use of a Bluetooth controller both exceed the minimum performance indicators. These properties allow for a load up to 100% of the quadruped's weight to be successfully transported on smooth surfaces and enable the control of the robot using both keyboard and game controllers.

Along with comparison to the performance criteria outlined in the scope of this project, the outcome of this project should also consider the economic, environmental and societal impacts it has. From an economic perspective, the total cost of the implementation remained under the allocated budget of $400. This demonstrates the cost-effectiveness of the solution, allowing for replication in similar educational environments.

Although the chosen material may introduce microplastics, the emphasis on modularity in design removes the need for numerous complex parts and facilitates the reuse of components. Furthermore, the energy efficiency of the project exceeds expectations, reducing its reliability on non-renewable energy sources.

From a social standpoint, the project poses minimal effects towards the wider community. The prototype does not include cameras, or store any form of data it collects, thereby eliminating the threat to privacy. Additionally, the default state of the robot is to remain stationary when no input is provided, helping precent collisions due to unintended movement. It is also unlikely that the project will result in job displacement due to its limited functional scope.

Given the success against the performance criteria, it can be concluded that the designed modular quadruped robot is capable of traversing uneven urban terrain. It effectively addresses the need of reconfigurability and reduces maintenance complexity through its integration of modular ideologies to achieve rapid assembly processes.

Chapter 6. Future Work

The main limitation of the quadruped robot designed in this project is its simplistic control system. The open-loop structure of this system prevents the robot from performing self-stabilisation and instead, relies upon the user to adjust limited gait parameters via a controller. This behaviour prevents fine tuning and real-time adjustment of the gait when traversing uneven terrain.

To solve this issue, two additions could be made. The first is to integrate the data collected by the IMU into a closed-loop control system using PID, Fuzzy, ZMP or a combination of these controllers to provide greater stability during motion and allow for self-adjustment of the robot.

Additionally, the modular structure of the quadruped could be leveraged to include an optical module. This module could contain RGBD cameras and a RasberryPi processing unit to provide greater perception of the robot's environment. The implementation of visual sensors also allows for object detection and automation to be applied, enabling self-navigation of the quadruped.

Another improvement to the designed system could be to further iterate the connection mechanism. In its current form the connection plate assembly requires the custom connector to be twist locked in place before it is joined with another module. This is because the connection faces are mirrored, requiring the custom connectors to be rotated in opposite directions to be removed. This design was intentional to prevent the connectors from coming loose during operation; however, it resulted in unintentional limitations. Due to the low tolerance of header pins, when multiple modules are connected side by side, any slight misalignment results in difficulty connecting the modules.

A possible solution to this mechanism is to allow the twist lock mechanism to be inserted with a rotation in either direction. This would allow the plates to be connected first, followed by the custom connecter, reducing the alignment process.

The use of stronger magnets for the connector would also be a welcome addition to this project, as although the connectors are able to support the weight of individual modules, they reach their connection strength limit under the full weight of the robot.

To improve the user experience with this robot, a server application could be created to allow for the observation of data collected by the onboard electronics, without a tether. Currently, observation of IMU and battery monitor data is limited to serial communication, requiring direct connection to a computer to view this data. Implementing a server application would enable the robot to address this limitation. Such applications can be hosted on the included ESP32 module by leveraging its open-source WIFI framework.

Finally, a digital twin in software such as MSC ADAMs would provide greater theoretical analysis of the designed robot, enabling it to be simulated in a variety of environments. The current simulation method which utilises SolidWorks, is limited by its inability to interact with custom control systems and instead, simulates motion using predefined motor equations. The use of MSC ADAMs would allow for the simulation to overcome these issues.

References

[1] M. Lu, B. Jing, H. Duan, and G. Gao, "Design of a Small Quadruped Robot with Parallel Legs," *Complexity,* vol. 2022, no. 1, p. 9663746, 2022, doi: https://doi.org/10.1155/2022/9663746.

[2] Q. Li, F. Cicirelli, A. Vinci, A. Guerrieri, W. Qi, and G. Fortino, "Quadruped Robots: Bridging Mechanical Design, Control, and Applications," *Robotics*, vol. 14, no. 5*,* doi: 10.3390/robotics14050057.

[3] J. Zhang, D. Zhang, and Z. Guo, "Research on quadruped robots: Review and Prospect," New York, NY, USA, 2023 2023: ACM, pp. 38-45, doi: 10.1145/3606843.3606850. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3606843.3606850

[4] Y. Fan, Z. Pei, C. Wang, M. Li, Z. Tang, and Q. Liu, "A Review of Quadruped Robots: Structure, Control, and Autonomous Motion," *Advanced Intelligent Systems,* vol. 6, no. 6, p. 2300783, 2024/06/01 2024, doi: https://doi.org/10.1002/aisy.202300783.

[5] Y. Zhong, R. Wang, H. Feng, and Y. Chen, "Analysis and research of quadruped robot's legs: A comprehensive review," *International Journal of Advanced Robotic Systems,* vol. 16, no. 3, p. 1729881419844148, 2019/05/01 2019, doi: 10.1177/1729881419844148.

[6] P. Biswal and P. K. Mohanty, "Development of quadruped walking robots: A review," *Ain Shams Engineering Journal,* vol. 12, no. 2, pp. 2017-2031, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447920302501.

[7] G. Bledt, M. J. Powell, B. Katz, J. D. Carlo, P. M. Wensing, and S. Kim, "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1-5 Oct. 2018 2018, pp. 2245-2252, doi: 10.1109/IROS.2018.8593885. [Online]. Available: https://doi.org/10.1109/IROS.2018.8593885

[8] J. Carpentier and P.-B. Wieber, "Recent Progress in Legged Robots Locomotion Control," *Current Robotics Reports,* vol. 2, no. 3, pp. 231-238, 2021/09/01 2021, doi: 10.1007/s43154-021-00059-0.

[9] A. Hamrani, R. M. Munim, M. Telusma, M. Dwayne, and L. and Lagos, "Smart quadruped robotics: a systematic review of design, control, sensing and perception," *Advanced Robotics,* vol. 39, no. 1, pp. 3-29, 2025/01/02 2025, doi: 10.1080/01691864.2024.2411684.

[10] A. H. Abdulwahab, A. Z. A. Mazlan, A. F. Hawary, and N. H. Hadi, "Quadruped robots mechanism, structural design, energy, gait, stability, and actuators: a review study," *International Journal of Mechanical Engineering and Robotics Research,* vol. 12, no. 6, pp. 385-395, 2023, doi: 10.18178/ijmerr.12.6.385-395.

[11] J. He and F. Gao, "Mechanism, actuation, perception, and control of highly dynamic multilegged robots: A review," *Chinese Journal of Mechanical Engineering,* vol. 33, pp. 1-30, 2020. [Online]. Available: https://cjme.springeropen.com/articles/10.1186/s10033-020-00485-9.

[12] H. Taheri and N. Mozayani, "A study on quadruped mobile robots," *Mechanism and Machine Theory,* vol. 190, p. 105448, 2023/12/01/ 2023, doi: https://doi.org/10.1016/j.mechmachtheory.2023.105448.

[13] H. Li, C. Qi, F. Gao, X. Chen, Y. Zhao, and Z. Chen, "Mechanism design and workspace analysis of a hexapod robot," *Mechanism and Machine Theory,* vol. 174, p. 104917, 2022/08/01/ 2022, doi: https://doi.org/10.1016/j.mechmachtheory.2022.104917.

[14] X. Zhou and S. Bi, "A survey of bio-inspired compliant legged robot designs," *Bioinspiration & Biomimetics,* vol. 7, no. 4, p. 041001, 2012/11/14 2012, doi: 10.1088/1748-3182/7/4/041001.

[15] N. Kau, A. Schultz, N. Ferrante, and P. Slade, "Stanford Doggo: An Open-Source, Quasi-Direct-Drive Quadruped," in *2019 International Conference on Robotics and Automation (ICRA)*, 20-24 May 2019 2019, pp. 6309-6315, doi: 10.1109/ICRA.2019.8794436. [Online]. Available: https://doi.org/10.1109/ICRA.2019.8794436

[16] G. Yadav, S. Jaiswal, and G. C. Nandi, *Generic Walking Trajectory Generation of Biped using Sinusoidal Function and Cubic Spline*. 2020, pp. 745-750.

[17] K. Y. Kim and J. H. Park, "Ellipse-based leg-trajectory generation for galloping quadruped robots," *Journal of Mechanical Science and Technology,* vol. 22, no. 11, pp. 2099-2106, 2008/11/01 2008, doi: 10.1007/s12206-008-0705-1.

[18] H. Chai *et al.*, "A survey of the development of quadruped robots: Joint configuration, dynamic locomotion control method and mobile manipulation approach," *Biomimetic Intelligence and Robotics,* vol. 2, no. 1, p. 100029, 2022/03/01/ 2022, doi: https://doi.org/10.1016/j.birob.2021.100029.

[19] K. Gilpin and D. Rus, "Modular Robot Systems," *IEEE Robotics & Automation Magazine,* vol. 17, no. 3, pp. 38-55, 2010, doi: 10.1109/MRA.2010.937859.

[20] A. Brunete, A. Ranganath, S. Segovia, J. P. de Frutos, M. Hernando, and E. Gambao, "Current trends in reconfigurable modular robots design," *International Journal of Advanced Robotic Systems,* vol. 14, no. 3, p. 1729881417710457, 2017/05/01 2017, doi: 10.1177/1729881417710457.

[21] J. Liu, X. Zhang, and G. Hao, "Survey on research and development of reconfigurable modular robots," *Advances in Mechanical Engineering,* vol. 8, no. 8, p. 1687814016659597, 2016/08/01 2016, doi: 10.1177/1687814016659597.

[22] A. Castano, A. Behar, and P. M. Will, "The Conro modules for reconfigurable robots," *IEEE/ASME Transactions on Mechatronics,* vol. 7, no. 4, pp. 403-409, 2002, doi: 10.1109/TMECH.2002.806233.

[23] E. Guizzo. (2012, 28/05/2012) Smart Pebble Robots Will Let You Duplicate Objects on the Fly. *IEEE Spectrum*. Available: https://spectrum.ieee.org/ai-weather-forecasting

[24] W. Saab, P. Racioppo, and P. Ben-Tzvi, "A review of coupling mechanism designs for modular reconfigurable robots," *Robotica,* vol. 37, no. 2, pp. 378-403, 2019, doi: 10.1017/S0263574718001066.

[25] C. Zhang, P. Zhu, Y. Lin, Z. Jiao, and J. Zou, "Modular Soft Robotics: Modular Units, Connection Mechanisms, and Applications," *Advanced Intelligent Systems,* vol. 2, no. 6, p. 1900166, 2020/06/01 2020, doi: https://doi.org/10.1002/aisy.201900166.

[26] J. Kim, T. Kang, D. Song, and S.-J. Yi, "Design and Control of a Open-Source, Low Cost, 3D Printed Dynamic Quadruped Robot," *Applied Sciences*, vol. 11, no. 9, doi: 10.3390/app11093762.

[27] H.-C. Zhuang, H.-B. Gao, and Z.-Q. Deng, "Analysis Method of Articulated Torque of Heavy-Duty Six-Legged Robot under Its Quadrangular Gait," *Applied Sciences*, vol. 6, no. 11, doi: 10.3390/app6110323.

[28] T. Demjen *et al.*, "Design of the five-bar linkage with singularity-free workspace," *Robotica,* vol. 41, no. 11, pp. 3361-3379, 2023, doi: 10.1017/S0263574723001042.

[29] H. D. Eckhardt, *Kinematic design of machines and mechanisms*. McGraw-Hill, 1998.

[30] S. O. Owoeye, F. Durodola, P. O. Adeniyi, I. Tolulope, and A. B. H. Abdullahi, "Design and development of a quadruped home surveillance robot," *IAES International Journal of Robotics and Automation,* vol. 13, no. 2, pp. 233-246, 2024. [Online]. Available: https://ijra.iaescore.com/index.php/IJRA/article/view/20658.

[31] S. E. Schoedel, A. J. Fuge, B. Kalita, and A. Leonessa, "Development of an Affordable and Modular 3D Printed Quadruped Robot," 2022. [Online]. Available: https://doi.org/10.1115/IMECE2022-95700.

[32] P. Stoffregen. "Teensy® 4.0 Development Board." https://www.pjrc.com/store/teensy40.html#tech (accessed 2/10/2025.

[33] C. F. L. Olvera and D. A. Flores-Hernandez, "Embedded System for Quadruped Robot in Mammalian Configuration," *IEEE Embedded Systems Letters,* pp. 1-1, 2025, doi: 10.1109/LES.2025.3541130.

[34] Esspressif, "ESP32-WROVER-E & ESP32-WROVER-IE Datasheet," 2025. [Online]. Available: https://documentation.espressif.com/esp32-wrover-e_esp32-wrover-ie_datasheet_en.pdf

[35] RasberryPI, "Raspberry Pi 4 Model B," 01/02/2025 2025. [Online]. Available: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf

[36] K. Hussain, Z. Omar, X. Wang, O. Orelaja, and M. Elnour, "Analysis and Research of Quadruped Robot's Actuators: A Review," *International Journal of Mechanical Engineering and Robotics Research,* vol. 1o, pp. 436-442, 08/01 2021, doi: 10.18178/ijmerr.10.8.436-442.

[37] M. Hutter *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, 2016: IEEE, pp. 38-44. [Online]. Available: https://doi.org/10.1109/IROS.2016.7758092. [Online]. Available: https://doi.org/10.1109/IROS.2016.7758092

[38] E. Prassler and A. Baroncelli, "2017 IEEE\/IFR Award for IERA [Industrial Activities]," *IEEE Robotics & Automation Magazine,* vol. 24, no. 3, pp. 8-11, 2017, doi: 10.1109/MRA.2017.2722203.

[39] G. Kenneally, A. De, and D. E. Koditschek, "Design Principles for a Family of Direct-Drive Legged Robots," *IEEE Robotics and Automation Letters,* vol. 1, no. 2, pp. 900-907, 2016, doi: 10.1109/LRA.2016.2528294.

[40] B. Shu, Z. Huang, W. Ren, Y. Wu, and H. Li, "Learning-Based Model Predictive Control for Legged Robots with Battery–Supercapacitor Hybrid Energy Storage System," *Applied Sciences*, vol. 15, no. 1, doi: 10.3390/app15010382.

[41] C. Comanescu, "Ensuring Safety and Reliability: An Overview of Lithium-Ion Battery Service Assessment," *Batteries*, vol. 11, no. 1, doi: 10.3390/batteries11010006.

[42] B. Xia, J. Fu, H. Zhu, Z. Song, Y. Jiang, and H. Lipson, "A Legged Soft Robot Platform for Dynamic Locomotion," in *2021 IEEE International Conference*

on Robotics and Automation (ICRA), 30 May-5 June 2021 2021, pp. 11812-11819, doi: 10.1109/ICRA48506.2021.9561018. [Online]. Available: https://doi.org/10.1109/ICRA48506.2021.9561018

[43] Y. Chen, J. E. Grezmak, N. M. Graf, and K. A. Daltorio, "Sideways crab-walking is faster and more efficient than forward walking for a hexapod robot," *Bioinspiration & Biomimetics,* vol. 17, no. 4, 046001, 11/05/2022 2022, doi: 10.1088/1748-3190/ac6847.

[44] Y. Shi, S. Li, M. Guo, Y. Yang, D. Xia, and X. Luo, "Structural Design, Simulation and Experiment of Quadruped Robot," *Applied Sciences*, vol. 11, no. 22*,* doi: 10.3390/app112210705.

[45] K. Mori *et al.*, "A Study of Trot Gait Control System of a Quadruped Robot Using IMU Sensor," in *2022 61st Annual Conference of the Society of Instrument and Control Engineers (SICE)*, 6-9 Sept. 2022 2022, pp. 849-854, doi: 10.23919/SICE56594.2022.9905832. [Online]. Available: https://doi.org/10.23919/SICE56594.2022.9905832

[46] F. Iida, G. Gomez, and R. Pfeifer, "Exploiting body dynamics for controlling a running quadruped robot," in *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, 18-20 July 2005 2005, pp. 229-235, doi: 10.1109/ICAR.2005.1507417.

[47] Y. Wang, "Quadruped Robot's Gait Planning and Kinematic Analysis," *Applied and Computational Engineering,* vol. 111, 29/11/2024 2024, doi: https://doi.org/10.54254/2755-2721/111/2024CH0116.

[48] K. L. S. Sharma, "6 - Automation Strategies," in *Overview of Industrial Process Automation (Second Edition)*, K. L. S. Sharma Ed.: Elsevier, 2017, pp. 53-74.

[49] A. A. Aldair, A. Al-Mayyahi, and W. Wang, "Design of a Stable an Intelligent Controller for a Quadruped Robot," *Journal of Electrical Engineering & Technology,* vol. 15, no. 2, pp. 817-832, 2020/03/01 2020, doi: 10.1007/s42835-019-00332-5.

[50] M. Li, Z. Liu, M. Wang, G. Pang, and H. Zhang, "Design of a Parallel Quadruped Robot Based on a Novel Intelligent Control System," *Applied Sciences*, vol. 12, no. 9*,* doi: 10.3390/app12094358.

[51] D. J. Blackman, "Control of Legged Robotic Systems: Substantiation of Gait Design, Multi-modal Behaviors, and Dynamic Scaling Theory in Practice," M.S., The Florida State University, United States -- Florida, 10840388, 2018. [Online]. Available: https://www.proquest.com/dissertations-theses/control-legged-robotic-systems-substantiation/docview/2121115072/se-2?accountid=10382

[52] H. Y. Wang, L. J. Chen, W. S. Yu, and P. C. Lin, "A Wheel to Leg Transformation Strategy in a Leg-Wheel Transformable Robot," in *2023 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, 28-30 June 2023 2023, pp. 293-298, doi: 10.1109/AIM46323.2023.10196156.

[53] Y. Zou, D. Kim, P. Norman, J. Espinosa, J.-C. Wang, and G. S. Virk, "Towards robot modularity — A review of international modularity standardization for service robots," *Robotics and Autonomous Systems,* vol. 148, p. 103943, 2022/02/01/ 2022, doi: https://doi.org/10.1016/j.robot.2021.103943.

[54] R. Michikawa *et al.*, "Development and verification of connectors for modular robots to complete practical tasks on the moon," *Artificial Life and Robotics,*

vol. 30, no. 2, pp. 198-207, 2025/05/01 2025, doi: 10.1007/s10015-024-00987-y.

[55] C. Parrott, T. J. Dodd, and R. Groß, "HiGen: A high-speed genderless mechanical connection mechanism with single-sided disconnect for self-reconfigurable modular robots," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 14-18 Sept. 2014 2014, pp. 3926-3932, doi: 10.1109/IROS.2014.6943114. [Online]. Available: https://doi.org/10.1109/IROS.2014.6943114

[56] J. Kim, A. Alspach, and K. Yamane, "Snapbot: A reconfigurable legged robot," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 24-28 Sept. 2017 2017, pp. 5861-5867, doi: 10.1109/IROS.2017.8206477. [Online]. Available: https://doi.org/10.1109/IROS.2017.8206477

[57] S. Farah, D. G. Anderson, and R. Langer, "Physical and mechanical properties of PLA, and their functions in widespread applications — A comprehensive review," *Advanced Drug Delivery Reviews,* vol. 107, pp. 367-392, 2016/12/15/ 2016, doi: https://doi.org/10.1016/j.addr.2016.06.012.

[58] G. Gao, F. Xu, J. Xu, and Z. Liu, "Study of Material Color Influences on Mechanical Characteristics of Fused Deposition Modeling Parts," *Materials*, vol. 15, no. 19*,* doi: 10.3390/ma15197039.

[59] I. Blanco, G. Cicala, G. Recca, and C. Tosto, "Specific Heat Capacity and Thermal Conductivity Measurements of PLA-Based 3D-Printed Parts with Milled Carbon Fiber Reinforcement," *Entropy*, vol. 24, no. 5*,* doi: 10.3390/e24050654.

[60] FeeTech, "PRODUCT SPECIFICATION FS5109R," 02/08/2021 2021. [Online]. Available: https://www.feetechrc.com/Data/feetechrc/upload/file/20210827/6376565881266889439933525.pdf

[61] J. J. Craig, *Introduction to robotics: mechanics and control, 3/E*, 3 ed. Pearson Education India, 2019.

[62] L. Erwin-Christian, C. Valentin, D. Tivadar, O. Alexandru, T. Elida-Gabriela, and S. Melania-Olivia, "Optimal Synthesis of Five-Bar Linkage Based on Singularity-Free Workspaces with Predefined Shapes," *Robotics*, vol. 13, no. 12*,* doi: 10.3390/robotics13120173.

[63] W. Voss, *A Comprehensible Guide to Servo Motor Sizing*. Greenfield, Massachusetts: Copperhill Technologies Corporation, 2007.

[64] B. Earl, "Adafruit PCA9685 16-Channel Servo Driver," Adafruit, 21/01/2025 2025. [Online]. Available: https://learn.adafruit.com/16-channel-pwm-servo-driver/pinouts

[65] V. C. Necula, D. Stamate, and G. Bucur, "SPIDER4LEGS MOBILE ROBOT DESIGN CONTROLLED BY THE ESP32 PLATFORM," *Romanian Journal of Petroleum & Gas Technology,* vol. 4, no. 1, pp. 251-258, 2023. [Online]. Available: https://journals.indexcopernicus.com/search/article?articleId=4470238.

[66] L. Clark, "Adafruit DC Power BFF," Adafruit, 08/03/2024 2024. [Online]. Available: https://learn.adafruit.com/adafruit-dc-power-bff

[67] A. Nayak, H. Seo, N. Gravish, and M. T. Tolley, "Burrowing and unburrowing in submerged granular media through fluidization and shape-change," (in English), *Frontiers in Robotics and AI,* Original Research vol. Volume 12 - 2025, 2025-July-31 2025, doi: 10.3389/frobt.2025.1546407.

[68] B. Siepert and K. Rembor, "Adafruit TDK InvenSense ICM-20948 9-DoF IMU," Adafruit, 22/01/2025 2025. [Online]. Available: https://learn.adafruit.com/adafruit-tdk-invensense-icm-20948-9-dof-imu
[69] R. Quesada, "esp-idf-arduino-bluepad32-template," 4.2.2 ed: Github, 2025.
[70] M. Amidon, K. Patterson, and G. Tewolde, "Accessible Control of a Robotic Arm for People With Physical Mobility Limitations," in *2024 IEEE Integrated STEM Education Conference (ISEC)*, 9-9 March 2024 2024, pp. 1-4, doi: 10.1109/ISEC61299.2024.10665259. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10665259
[71] Adafruit, "Adafruit_INA260," 1.5.3 ed: Github, 2023.
[72] Adafruit, "Adafruit_ICM20X," 2.0.7 ed: Github, 2023.
[73] Adafruit, "Adafruit-PWM-Servo-Driver-Library," 3.0.2 ed: Github, 2024.
[74] E. Batt, "MORPH_ESP_IDF," ed. Github: Curtin University, 2025.

Appendices

Appendix A: Derivation of the Leg Mechanism Geometries

The following MATLAB script applies the process outlined in section 3.1 to derive the lengths of a five-bar linkage utilised as the leg mechanism in this project.

Set Parameters

```
% define indexes for x and y coordinates
x = 1; % [x,y] = [1,2]
y = 2; % [x,y] = [1,2]


% define gait parameters
duty_cycle = 0.25; % time spent in swing phase
total_cycle = 1.0; % time in seconds to complete a full step
num_points = 100; % number of points in curve
step_length = 100; % mm
step_height = 50; % mm
step_offset = 50; % mm
% define step to be symmetrical about the y-axis
step_start = [-step_length/2, step_height+step_offset]; % starting
(lifting) point
step_end = [step_length/2, step_height+step_offset]; % ending (landing)
point


x = 1; % index for x component
y = 2; % index for y component


% Dexterous workspace
m_1 = [step_start(x), step_start(y)-step_height]; % in mm from the
midpoint of l_1
m_2= [step_end(x), step_end(y)-step_height]; % in mm from the midpoint
of l_1
m_3 = [step_end(x), step_start(y)]; % in mm from the midpoint of l_1
m_4 = [step_start(x), step_end(y)]; % in mm from the midpoint of l_1


% specify u_23 and l_1
u_23 = deg2rad(20); % rad
l_1 = 50; % mm
```

Synthesis of 5-Bar Linkage

```
% applying equation 25 from [21]
numerator = m_1(y)^2-sqrt(m_1(y)^4-
((m_3(x)+l_1/2)^2+m_3(y)^2)*(m_1(y)^2)*(1-cos(u_23)^2));
denominator = ((m_3(x) + l_1/2)^2 + m_3(y)^2) * (1 - cos(u_23));
k = sqrt( numerator / denominator )


l_1 = l_1 % predefined
l_2 = 0.5 * ( k*sqrt( (m_3(x)+l_1/2)^2 + m_3(y)^2 ) - m_1(y)/k ) %
applying equation 21 from [21]
l_3 = 0.5 * ( k*sqrt( (m_3(x)+l_1/2)^2 + m_3(y)^2 ) + m_1(y)/k ) %
applying equation 22 from [21]
l_4 = l_3 % mechanism is symmetrical
l_5 = l_2 % mechanism is symmetrical
```

Output

```
k = 1.1302
l_1 = 50
l_2 = 48.5169
l_3 = 92.7572
l_4 = 92.7572
l_5 = 48.5169
```

5-Bar Linkage Visualisation

```
% Define Actuator Positions
a_0 = [-l_1/2, 0];
e_0 = [l_1/2, 0];
% Define End Effector Position
x_m = -50;
y_m = 75;
% Perform Inverse Kinematics for given Position
[phi_2,phi_3,phi_4,phi_5,b_0,d_0,m_0] = solveIK(x_m, y_m, l_1, l_2,
l_3, l_4, l_5);
% Plot links
figure;
hold on;
grid on;
xlabel('X [mm]');
ylabel('Y [mm]');
title('Five-bar Linkage IK Visualization');
xlim([m_1(x)*2, m_2(x)*2]);
ylim([-m_4(y)*1.5, m_1(y)*1]);
% Plot Axis of Symmetry
plot([0, 0], [-m_4(y)*1.5, m_1(y)*1], 'm--', 'LineWidth', 1)
% Plot Origin
```

```
plot(0, 0, 'mo', 'MarkerSize', 10, 'MarkerFaceColor', 'm'); % Foot
% Plot Minimum Transmition angle
plot_u_23 = linspace(phi_2, phi_2 + mod(pi - phi_2 + phi_3, 2*pi), 50);
plot(b_0(x) - 20*cos(plot_u_23), -b_0(y) + 20*sin(plot_u_23), 'm-',
'LineWidth', 1)
% Plot workspace
plot([m_1(x), m_2(x), m_3(x), m_4(x), m_1(x)], ...
     -[m_1(y), m_2(y), m_3(y), m_4(y), m_1(y)], ...
     'r-o', 'LineWidth', 2);
% Plot Leg Mechanism
plot([a_0(x), e_0(x)], -[a_0(y), e_0(y)], 'ks-', 'LineWidth', 2); %
Ground Link
plot([a_0(x), b_0(x)], -[a_0(y), b_0(y)], 'b-o', 'LineWidth', 2); %
Link 2
plot([b_0(x), m_0(x)], -[b_0(y), m_0(y)], 'g-o', 'LineWidth', 2); %
Link 3
plot([d_0(x), m_0(x)], -[d_0(y), m_0(y)], 'g-o', 'LineWidth', 2); %
Link 4
plot([e_0(x), d_0(x)], -[e_0(y), d_0(y)], 'b-o', 'LineWidth', 2); %
Link 5
plot(m_0(x), -m_0(y), 'ko', 'MarkerSize', 10, 'MarkerFaceColor', 'k');
% Foot
legend(["Axis of Symmetry", "Origin", "Transmission Angle (u_m_i_n))",
...
    "Workspace (M_1, M_2, M_3, M_4)", "Ground Link (l_1)", "Upper Link
(l_2)", ...
    "Lower Link (l_3)", "Lower Link (l_4)", "Upper Link (l_5)",
"Foot"]);
hold off;
```

Output:

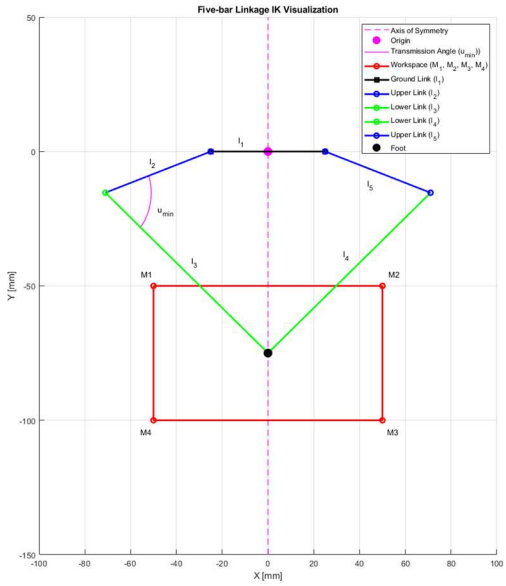

Figure 77: Derivation of Leg Parameters MATLAB Output.

Appendix B: Design of the Foot Trajectory

The following MATLAB script applies the process outlined in section 3.2.1 to generate the discrete increments of the desired foot trajectory. This script is designed to be run using the parameters defined in Appendix A. Portions of the script can be uncommented to display the workspace around the trajectory.

Design of Gait Trajectory

```
% preallocate arrays
x_gait = zeros(size(num_points));
y_gait = zeros(size(num_points));


% calculate swing phase
for i = 1:num_points
    % apply trajectory forumla
    [x_gait(i), y_gait(i)] = nextStep(step_length, step_height, i,
duty_cycle, total_cycle, num_points, step_start(x), step_offset,
false);
end
% plot trajectory
% plot inverted to suit IK function. reflipped to correct orientation
for
% display
figure;
hold on;
% xlim([m_1(x)*2, m_2(x)*2]); % Used to Plot Workspace
% ylim([-m_4(y)*1.5, m_1(y)*1]); % Used to Plot Workspace
% Plot workspace
% plot([m_1(x), m_2(x), m_3(x), m_4(x), m_1(x)], ...
     % -[m_1(y), m_2(y), m_3(y), m_4(y), m_1(y)], ...
     % 'r-o', 'LineWidth', 2);
% Plot Discrete intervals
plot(x_gait, -y_gait, 'bo', 'MarkerSize', 5, 'MarkerFaceColor', 'b');
% Plot Lifting and Landing point
plot(step_start(x), -step_start(y), 'go', 'MarkerSize', 10,
'MarkerFaceColor', 'g');
plot(step_end(x), -step_end(y), 'ro', 'MarkerSize', 10,
'MarkerFaceColor', 'm');
xlabel('x position (mm)');
ylabel('y position (mm)');
title('Gait Trajectory');
grid on;
axis equal;
legend('Gait trajectory', 'Lifting point', 'Landing point');
```

Gait Trajectory

```
function [g_x, g_y] = nextStep(g_l, g_h, i, lmda, t_s, resolution,
x_bound, y_bound, isCycloidal)
    % sigma_ = 2*pi*(i-resolution*lmda-1)/((1-lmda)*resolution*t_s);
    sigma = 2*pi*(i-1)/(lmda*resolution*t_s);
    dx = g_l / (resolution*(1-lmda)-1); % calculate step size during
return phase
    if isCycloidal
        if i <= resolution*lmda % Check if swing phase
            % calculate x and y coordinates for an elliptical swing
phase
            g_x = ((g_l * (sigma - sin(sigma)) / (2 * pi))) + (-g_l/2);
            g_y = -((g_h * (1 - cos(sigma)) / 2)) + y_bound+g_h;
        else % Otherwise return phase
            % calculate x and y coordinates for a horizontal return
phase
            g_x = g_l/2 - (i - lmda*resolution -1) * dx;
            g_y = y_bound + g_h;
        end
    else
        if i <= resolution*lmda % Check if swing phase
            g_x = (g_l*sigma) / (2*pi) - g_l/2;
            g_y = -g_h * sin(g_x*pi/g_l + pi/2) + y_bound+g_h;
        else
            g_x = g_l/2 - (i - lmda*resolution -1) * dx;
            g_y = y_bound + g_h;
            % g_x = (g_l*sigma_) / (2*pi) - g_l/2;
            % g_y = g_h * (1-h_rto) * sin(g_x*pi/g_l + pi/2) +
y_bound+g_h*h_rto;
        end
    end
end
```

Outputs:

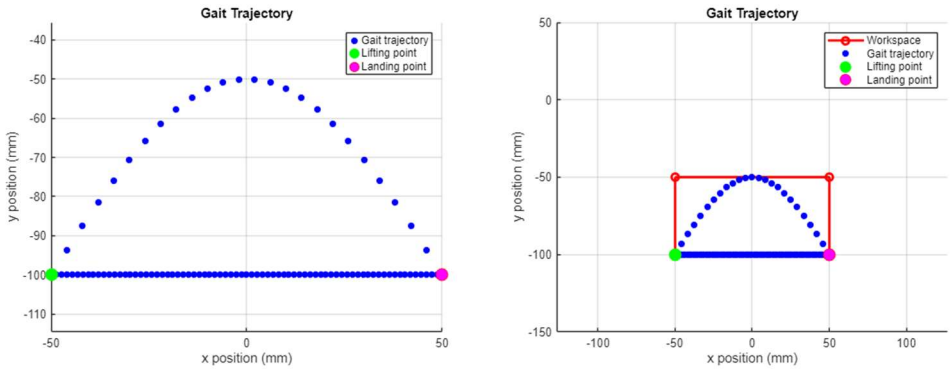

Figure 78: Design of Foot Trajectory MATLAB Output.

Appendix C: Position, Velocity and Acceleration Analysis

The following MATLAB script applies the process outlined in section 3.7.3 to generate the surfaces depicted in this analysis. This script is designed to be run using the parameters defined in Appendix A.

Forward Kinematic Analysis

Position Analysis

```
% creating a grid of possible actuator angles within their bounds
[theta_12,theta_15]=meshgrid(linspace(pi/2,pi*3/4,50),linspace(pi/2,pi*
3/4,50));
% using geometry to calculate the x and y component of l_s for all grid
% points
l_s_x = -l_5*cos(theta_15) + l_1 - l_2*cos(theta_12);
l_s_y = -l_5*sin(theta_15) - l_2*sin(theta_12);
% calculating the magnitude and direction of l_s
l_s = sqrt(l_s_x.^2 + l_s_y.^2);
theta_s = (atan(l_s_y./l_s_x)+pi);
% rearranging equations 7.26 and 7.27
theta_13 = -acos((l_4^2 - l_s.^2 -l_3^2)./(2*l_s*l_3)) + theta_s; %
theta_14 = acos((l_3^2 - l_s.^2 -l_4^2)./(2*l_s*l_4)) + theta_s;
% Plotting theta_13 surface
figure;
subplot(1, 2, 1);
surf(theta_12, theta_15, theta_13);
xlabel('\theta_12 (rad)');
ylabel('\theta_15 (rad)');
zlabel('\theta_13 (rad)');
grid on;
% Plotting theta_14 surface
subplot(1, 2, 2);
surf(theta_12, theta_15, theta_14);
xlabel('\theta_12 (rad)');
ylabel('\theta_15 (rad)');
zlabel('\theta_14 (rad)');
grid on;
sgtitle('Surface Plot of \theta_13 and \theta_14 w.r.t. \theta_12 and
\theta_15');
```

Velocity Analysis

```
% conduct velocity analysis from an arbitrary set position
phi_2 = deg2rad(160);
phi_5 = deg2rad(60);
% define vector angles used in loop analysis using link geometry
theta_11 = pi;
theta_12 = phi_2;
theta_15 = pi + phi_5;
% using geometry to calculate the x and y components of l_s
l_s_x = -l_5*cos(theta_15) + l_1 - l_2*cos(theta_12);
l_s_y = -l_5*sin(theta_15) - l_2*sin(theta_12);
% calculating the magnitude and direction of l_s
l_s = sqrt(l_s_x.^2 + l_s_y.^2);
theta_s = (atan(l_s_y/l_s_x)+pi);
% rearranging equations 7.26 and 7.27
theta_13 = -acos((l_4^2 - l_s^2 -l_3^2)/(2*l_s*l_3)) + theta_s;
theta_14 = acos((l_3^2 - l_s^2 -l_4^2)/(2*l_s*l_4)) + theta_s;
% Velocity analysis from set position
syms  omega_13 omega_14 % define as symbols for the moment
% define maximum actuator speed using preliminary selection
max_actuator_speed = 50*2*pi/60; % as per [60]
% creating a grid of possible actuator speeds within their bounds
[omega_12, omega_15] = meshgrid(linspace(-max_actuator_speed, ...
 max_actuator_speed,50),linspace(-
max_actuator_speed,max_actuator_speed,50));
% equations 7.30 and 7.31 in [22]
eq1 = omega_12*l_2*cos(theta_12) + omega_13*l_3*cos(theta_13 ...
    ) + omega_14*l_4*cos(theta_14) + omega_15*l_5*cos(theta_15) == 0;
eq2 = omega_12*l_2*sin(theta_12) + omega_13*l_3*sin(theta_13 ...
    ) + omega_14*l_4*sin(theta_14) + omega_15*l_5*sin(theta_15) == 0;
% solving equations using a matrix method
% moving known constants to the other side of the equation
X = omega_12*l_2*cos(theta_12) + omega_15*l_5*cos(theta_15);
Y = omega_12*l_2*sin(theta_12) + omega_15*l_5*sin(theta_15);
% defining coefficient matrices
a_11 = l_3*cos(theta_13);
a_12 = l_4*cos(theta_14);
a_21 = l_3*sin(theta_13);
a_22 = l_4*sin(theta_14);
% solving the matrix equation
A = [a_11, a_12; a_21, a_22];
B = [X(:)'; Y(:)'];
O = A\B;
% flattening the results for ploting
omega_13 = reshape(O(1, :), size(omega_12));
omega_14 = reshape(O(2, :), size(omega_12));
% Plotting omega_13 surface
figure;
hold on;
subplot(1, 2, 1);
```

```
surf(omega_12, omega_15, omega_13);
xlabel('\omega_12 (rad/s)');
ylabel('\omega_15 (rad/s)');
zlabel('\omega_13 (rad/s)');
view([-230 30.00])
% Plotting omega_14 surface
subplot(1, 2, 2);
surf(omega_12, omega_15, omega_14);
xlabel('\omega_12 (rad/s)');
ylabel('\omega_15 (rad/s)');
zlabel('\omega_14 (rad/s)');
grid on;
sgtitle('Surface Plot of \omega_13 and \omega_14 w.r.t. \omega_12 and
\omega_15');
view([-230 30.00])
```

Acceleration Analysis

```
% conduct velocity analysis from an arbitrary set veloxity
max_actuator_speed = 50*2*pi/60; % as per [60]
omega_12 = max_actuator_speed;
omega_15 = max_actuator_speed;
% equations 7.30 and 7.31
eq1 = omega_12*l_2*cos(theta_12) + omega_13*l_3*cos(theta_13 ...
    ) + omega_14*l_4*cos(theta_14) + omega_15*l_5*cos(theta_15) == 0;
eq2 = omega_12*l_2*sin(theta_12) + omega_13*l_3*sin(theta_13 ...
    ) + omega_14*l_4*sin(theta_14) + omega_15*l_5*sin(theta_15) == 0;
% solving equations using a matrix method
% Moving known constants to the other side of the equation
X = omega_12*l_2*cos(theta_12) + omega_15*l_5*cos(theta_15);
Y = omega_12*l_2*sin(theta_12) + omega_15*l_5*sin(theta_15);
% defining coefficient matrices
a_11 = l_3*cos(theta_13);
a_12 = l_4*cos(theta_14);
a_21 = l_3*sin(theta_13);
a_22 = l_4*sin(theta_14);
% solving the matrix equation
A = [a_11, a_12; a_21, a_22];
B = [X; Y];
O = A\B;
% obtaining the calculated velocities
omega_13 = O(1);
omega_14 = O(2);
% Acceleration analysis from set velocity
syms  alpha_13 alpha_14 % define as symbols for the moment
% defining arbitrary maximum acceleration
max_actuator_acceleration = 1; % rad/s^2
% creating a grid of possible actuator accelerations within their
bounds
```

```
[alpha_12, alpha_15] = meshgrid(linspace(-max_actuator_acceleration,
...
    max_actuator_acceleration, 50), linspace(-
max_actuator_acceleration, ...
    max_actuator_acceleration, 50));
% equations 7.38 and 7.39
eq1 = alpha_12*l_2*sin(theta_12) + alpha_13*l_3*sin(theta_13 ...
    ) + alpha_14*l_4*sin(theta_14) + alpha_15*l_5*sin(theta_15 ...
    ) + omega_12^2*l_2*cos(theta_12) + omega_13^2*l_3*cos(theta_13 ...
    ) + omega_14^2*l_4*cos(theta_14) + omega_15^2*l_5*cos(theta_15) ==
0;
eq2 = alpha_12*l_2*cos(theta_12) + alpha_13*l_3*cos(theta_13 ...
    ) + alpha_14*l_4*cos(theta_14) + alpha_15*l_5*cos(theta_15 ...
    ) - omega_12^2*l_2*sin(theta_12) - omega_13^2*l_3*sin(theta_13 ...
    ) - omega_14^2*l_4*sin(theta_14) - omega_15^2*l_5*sin(theta_15) ==
0;
% solving equations using a matrix method
% moving known constants to the other side of the equation
X = alpha_12*l_2*sin(theta_12) + alpha_15*l_5*sin(theta_15 ...
    ) - omega_12^2*l_2*cos(theta_12) - omega_13^2*l_3*cos(theta_13 ...
    ) - omega_14^2*l_4*cos(theta_14) - omega_15^2*l_5*cos(theta_15);
Y = alpha_12*l_2*cos(theta_12) + alpha_15*l_5*cos(theta_15 ...
    ) - omega_12^2*l_2*sin(theta_12) - omega_13^2*l_3*sin(theta_13 ...
    ) - omega_14^2*l_4*sin(theta_14) - omega_15^2*l_5*sin(theta_15);
% defining coefficient matrices
a_11 = l_3*sin(theta_13);
a_12 = l_4*sin(theta_14);
a_21 = l_3*cos(theta_13);
a_22 = l_4*cos(theta_14);
% solving the matrix equation
A = [a_11, a_12; a_21, a_22];
B = [X(:)'; Y(:)'];
O = A\B;
% flattening the results for ploting
alpha_13 = reshape(O(1, :), size(alpha_12));
alpha_14 = reshape(O(2, :), size(alpha_12));
% Plotting alpha_13 surface
figure;
subplot(1, 2, 1);
surf(alpha_12, alpha_15, alpha_13);
xlabel('\alpha_12 (rad/s^2)');
ylabel('\alpha_15 (rad/s^2)');
zlabel('\alpha_13 (rad/s^2)');
grid on;
% Plotting alpha_14 surface
subplot(1, 2, 2);
surf(alpha_12, alpha_15, alpha_14);
xlabel('\alpha_12 (rad/s^2)');
ylabel('\alpha_15 (rad/s^2)');
zlabel('\alpha_14 (rad/s^2)');
grid on;
```

```
sgtitle('Surface Plot of \alpha_13 and \alpha_14 w.r.t. \alpha_12 and
\alpha_15');
```

Outputs:

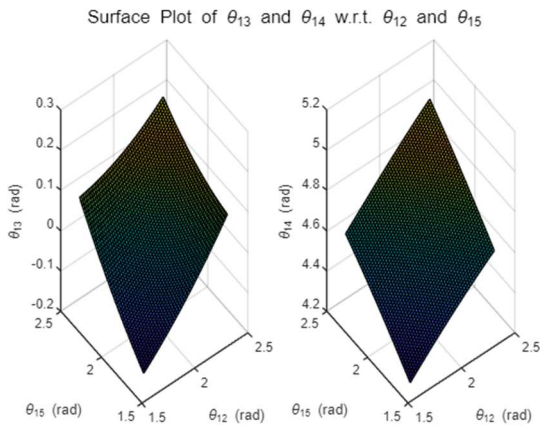

Figure 79: Position Analysis MATLAB Output.

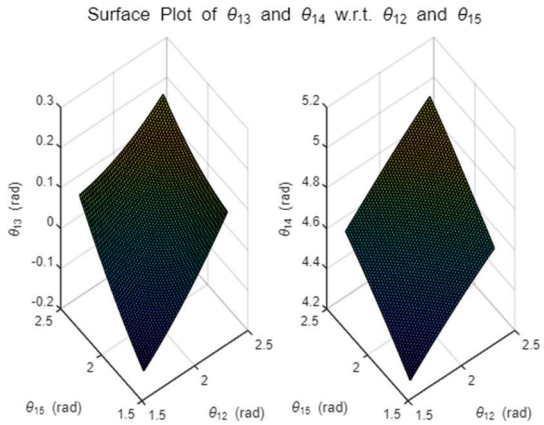

Figure 80 Velocity Analysis MATLAB Output.

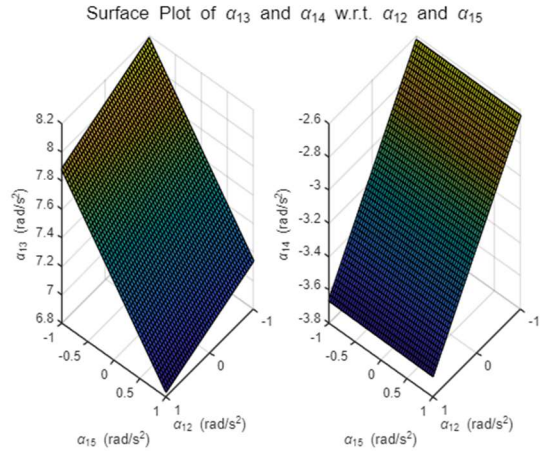

Figure 81: Acceleration Analysis MATLAB Output.

Appendix D: Static Approximation of Dynamic Behaviour

The following MATLAB script applies the process outlined in section 3.8.1 to approximate the dynamic behaviour of the leg mechanism over a single gait cycle. This script is designed to be run using the parameters defined in Appendix A and the trajectory defined in Appendix B.

Static Approximation of Dynamic Behaviour

```
w = 3.499/2; %kg
g = 9.81; %m/s^2
f = w*g; % combined self weight and external load of 1 body weight
u_s = 0.6;
n = 2.564; % gait duration
t = linspace(0, n, length(x_gait));
d_t = n/length(x_gait);
% static force analysis over gait
% preallocate arrays
t_2_array = zeros(1, length(x_gait));
t_5_array = zeros(1, length(x_gait));
f_3_array = zeros(1, length(x_gait));
f_4_array = zeros(1, length(x_gait));
phi_2_array = zeros(1, length(x_gait));
phi_5_array = zeros(1, length(x_gait));
phi2_dot_array = zeros(1, length(x_gait));
phi5_dot_array = zeros(1, length(x_gait));
phi2_ddot_array = zeros(1, length(x_gait));
phi5_ddot_array = zeros(1, length(x_gait));
% Loop over trajectory points
for i = 1:length(x_gait)
    % Solve IK
    [phi_2, phi_3, phi_4, phi_5, ~, ~, ~] = solveIK(x_gait(i),
y_gait(i), l_1, l_2, l_3, l_4, l_5);
    if phi_5 > pi
        phi_5 = phi_5 - 2*pi;
    end
    phi_2_array(i) = phi_2;
    phi_5_array(i) = phi_5;
    % Calculate derivatives
    if i > 2
        % For i > 2, we can safely use the previous values in the
arrays
        phi2_dot = (phi_2 - phi_2_array(i-1)) / d_t;
        phi2_ddot = (phi_2 - 2 * phi_2_array(i-1) + phi_2_array(i-2)) /
(d_t^2);
```

```
        phi5_dot = (phi_5 - phi_5_array(i-1)) / d_t;
        phi5_ddot = (phi_5 - 2 * phi_5_array(i-1) + phi_5_array(i-2)) /
(d_t^2);
    elseif i > 1
        % For i = 2, we have only one previous value
        phi2_dot = (phi_2 - phi_2_array(end-1)) / d_t;
        phi2_ddot = 0;  % No acceleration information yet

        phi5_dot = (phi_5 - phi_5_array(end-1)) / d_t;
        phi5_ddot = 0;
    else
        % For i = 1, there is no previous data
        phi2_dot = 0;
        phi2_ddot = 0;

        phi5_dot = 0;
        phi5_ddot = 0;
    end
    % if abs(phi5_ddot) > 100
    %     phi5_ddot = 0;
    % end
    %
    % if abs(phi2_ddot) > 100
    %     phi2_ddot = 0;
    % end
    % Store derivatives
    phi2_dot_array(i) = phi2_dot;
    phi2_ddot_array(i) = phi2_ddot;
    phi5_dot_array(i) = phi5_dot;
    phi5_ddot_array(i) = phi5_ddot;
    % Solve Static Forces
    [t_2, t_5, f_3, f_4] = staticForces(f, u_s, l_2, l_5, phi_2, phi_3,
phi_4, phi_5, duty_cycle, length(x_gait), i);
    % Store results
    t_2_array(i) = t_2;
    t_5_array(i) = t_5;
    f_3_array(i) = f_3;
    f_4_array(i) = f_4;
end
for i = 1:2
    [phi_2, phi_3, phi_4, phi_5, ~, ~, ~] = solveIK(x_gait(i),
y_gait(i), l_1, l_2, l_3, l_4, l_5);
    if phi_5 > pi
        phi_5 = phi_5 - 2*pi;
    end
    phi_2_array(i) = phi_2;
    phi_5_array(i) = phi_5;

    v_i = mod(i-2, length(x_gait))+1;
    a_i = mod(i-3, length(x_gait))+1;
    phi2_dot_array(i) = (phi_2 - phi_2_array(v_i)) / d_t;
```

```
    phi2_ddot = (phi_2 - 2 * phi_2_array(v_i) + phi_2_array(a_i)) /
(d_t^2);
    phi5_dot_array(i) = (phi_5 - phi_5_array(v_i)) / d_t;
    phi5_ddot = (phi_5 - 2 * phi_5_array(v_i) + phi_5_array(a_i)) /
(d_t^2);
    % if abs(phi5_ddot) > 100
    %     phi5_ddot = 0;
    % end
    %
    % if abs(phi2_ddot) > 100
    %     phi2_ddot = 0;
    % end
    phi2_ddot_array(i) = phi2_ddot;
    phi5_ddot_array(i) = phi5_ddot;
end
% Plot Displacement
figure;
hold on;
grid on;
xlabel('Time [s]');
ylabel('Angle [rad]');
title('Displacement over Time');
plot(t, phi_2_array, 'b', 'LineWidth', 2);
plot(t, phi_5_array, 'r', 'LineWidth', 2);
legend('\phi_2', '\phi_5');
hold off;
% Plot Velocity
figure;
hold on;
grid on;
xlabel('Time [s]');
ylabel('Angular Velocity [rad/s]');
title('Velocity over Time');
plot(t, phi2_dot_array, 'b-', 'LineWidth', 2);
plot(t, phi5_dot_array, 'r-', 'LineWidth', 2);
legend('\phi_2 dot', '\phi_5 dot');
hold off;
% Plot Acceleration
figure;
hold on;
grid on;
xlabel('Time [s]');
ylabel('Angular Acceleration [rad/s^2]');
title('Acceleration');
plot(t, phi2_ddot_array, 'b-', 'LineWidth', 2);
plot(t, phi5_ddot_array, 'r-', 'LineWidth', 2);
legend('\phi_2 ddot', '\phi_5 ddot');
hold off;
% Plot Torque
figure;
subplot(1, 2, 1);
```

```
hold on;
grid on;
xlabel('Time [s]');
ylabel('Torque [Nm]');
plot(t, t_2_array, 'r', 'LineWidth', 2);
plot(t, t_5_array, 'b', 'LineWidth', 2);
legend('t_2', 't_5');
hold off;
% Plot Force
subplot(1, 2, 2);
hold on;
grid on;
xlabel('Time [s]');
ylabel('Force [N]');
plot(t, f_3_array, 'k', 'LineWidth', 2);
plot(t, f_4_array, 'g', 'LineWidth', 2);
sgtitle('Torque at Actuators and Force at Applied to Links Over Gait
Cycle');
legend('f_3', 'f_4');
hold off;
```

Determination of Torques and Forces

```
function [t_2, t_5, f_3, f_4] = staticForces(f, u_s, l_2, l_5, phi_2,
phi_3, phi_4, phi_5, lambda, resolution, i)

    if (i > resolution*lambda)
        u_s = 0;
    end
    f_3 = f*cos(pi/2 - phi_3)-f*u_s*cos(phi_3);
    f_4 = f*cos(phi_4 - pi/2)+f*u_s*cos(180 - phi_4);
    % check bounds to ensure correct perpendicular distance is
calculated
    if phi_3 > (phi_2 - pi/2)
        t_2 = l_2*f_3*cos(phi_3 - phi_2 + pi/2)/1000; % torque applied
to link 2
    else
        t_2 = l_2*f_3*cos(phi_2 - phi_3 - pi/2)/1000; % torque applied
to link 2
    end
    % check bounds to ensure correct perpendicular distance is
calculated
    if phi_4 > (phi_5 + pi/2)
        t_5 = l_5*f_4*cos(phi_4 - phi_5 - pi/2)/1000; % torque applied
to link 5
    else
        t_5 = l_5*f_4*cos(phi_5 - phi_4 + pi/2)/1000; % torque applied
to link 5
    end
end
```

Inverse Kinematics

```
% takes in the coordinates of the end effector and the link lengths
% returns the angles of the links and the coordinates of the passive
joints
% including the manipulator
function [phi_2,phi_3,phi_4,phi_5,b_0,d_0,m_0] = solveIK(x_m, y_m, l_1,
l_2, l_3, l_4, l_5)
    % define the coordinates of each actuated joint
    a_0 = [-l_1/2, 0];
    e_0 = [l_1/2, 0];
    x = 1; % index for x component
    y = 2; % index for y component
    % calculate coefficients using equations 11 and 12 in [21]
    a_2 = -l_2*(2*x_m + l_1);
    b_2 = -2*l_2*y_m;
    c_2 = x_m^2 + y_m^2 + (l_1^2)/4 + l_2^2 - l_3^2 + l_1*x_m;

    a_5 = -l_5*(2*x_m - l_1);
    b_5 = -2*l_5*y_m;
    c_5 = x_m^2 + y_m^2 + (l_1^2)/4 + l_5^2 - l_4^2 - l_1*x_m;

    % solve for the angles of each actuated joint phi2, phi5
    % quadratic formula produces two solutions

    phi_2_n = mod(2 * atan((b_2-sqrt(a_2^2+b_2^2-c_2^2))/(a_2-c_2)),
2*pi);
    phi_2_p = mod(2 * atan((b_2+sqrt(a_2^2+b_2^2-c_2^2))/(a_2-c_2)),
2*pi);

    phi_5_n = mod(2 * atan((b_5-sqrt(a_5^2+b_5^2-c_5^2))/(a_5-c_5)),
2*pi);
    phi_5_p = mod(2 * atan((b_5+sqrt(a_5^2+b_5^2-c_5^2))/(a_5-c_5)),
2*pi);

    % check bounds to select the correct solution
    if phi_2_n <= 3/2*pi && phi_2_n >= pi/2
        phi_2 = phi_2_n;
    else
        phi_2 = phi_2_p;
    end
    % actuator for link 5 can move from 0 to 90 degrees or 270 to 360
degrees
    if phi_5_n <= pi/2 && phi_5_n >= 0 || ...
        phi_5_n <= 2*pi && phi_5_n >= 3/2*pi
        phi_5 = phi_5_n;
    else
        phi_5 = phi_5_p;
    end
```

```
    % compute the coordinates of the joints using geometry
    b_0 = a_0 + l_2 * [cos(phi_2), sin(phi_2)];
    d_0 = e_0 + l_5 * [cos(phi_5), sin(phi_5)];
    m_0 = [x_m, y_m];
    % compute joint angles of passive joints using geometry
    phi_3 = atan2(m_0(y)-b_0(y), m_0(x) - b_0(x));
    phi_4 = pi - atan2(m_0(y) - d_0(y), d_0(x) - m_0(x));
end
```

Outputs:

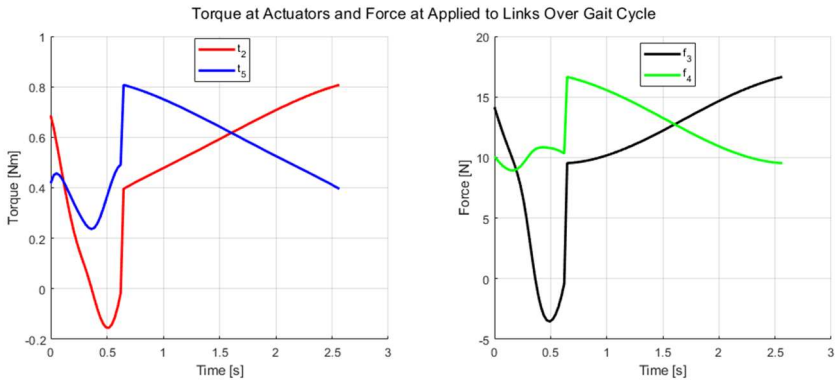

Figure 82: Static Approximation of Torques and Forces MATLAB Output.

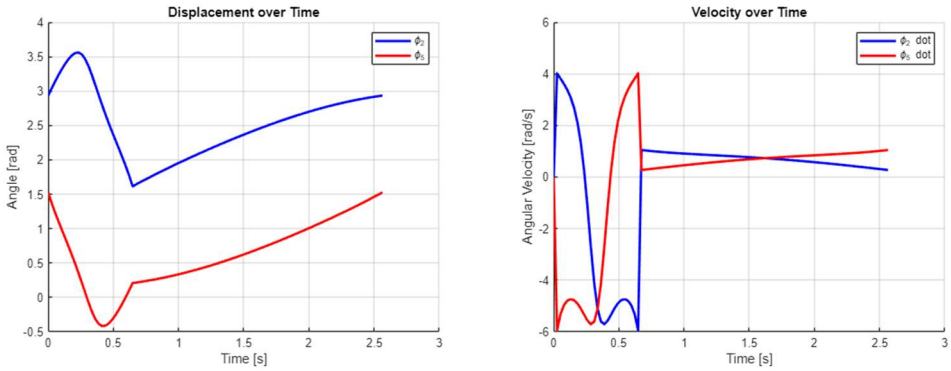

Figure 83: Static Approximation of Displacement and Velocity MATLAB Output.

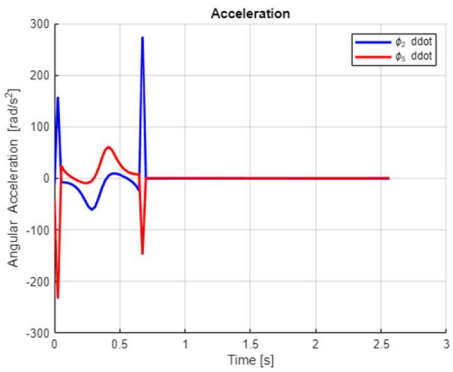

Figure 84: Static Approximation of Acceleration MATLAB Output.

Appendix E: Control System Source Code

The following C++ code outlines the control system employed in this project. This code does not include the Bluepad32 structure, however, it is designed to be taken and implemented into the provided Bluepad32 template, [69], for external use. The full source code is available online at [74] or via the link below.

Source Code Repository: https://github.com/Marradong/MORPH_ESP_IDF

Sketch.cpp

```
#include <Wire.h>
#include "batterymonitor.h"
#include "controller.h"
#include "imu.h"
#include "leg.h"
#include "servodriver.h"

BLEController blecontroller;
BatteryMonitor batterymonitor;
IMU imu;
ServoDriver servodriver;

uint8_t STATE = STOPPED;

Leg legFL(servodriver, SERVO_FLF, SERVO_FLB, true, LAMBDA*RESOLUTION);
Leg legFR(servodriver, SERVO_FRF, SERVO_FRB, true,
3*LAMBDA*RESOLUTION);
Leg legBL(servodriver, SERVO_BLF, SERVO_BLB, false, 0);
Leg legBR(servodriver, SERVO_BRF, SERVO_BRB, false,
2*LAMBDA*RESOLUTION);

BLEController::BLEControllerData ctrlData;
IMU::IMUData imuData;
BatteryMonitor::BatteryData batteryData;

void home() {
  legFL.Home();
  legFR.Home();
  legBL.Home();
  legBR.Home();
  delay(500);
}
```

```
void forward() {
  legFL.Forward();
  legFR.Forward();
  legBL.Forward();
  legBR.Forward();
}

void reverse() {
  legFL.Reverse();
  legFR.Reverse();
  legBL.Reverse();
  legBR.Reverse();
}

void left() {
  legFL.Turn(STEP_SHORT);
  legFR.Turn(STEP_LONG);
  legBL.Turn(STEP_SHORT);
  legBR.Turn(STEP_LONG);
}

void right() {
  legFL.Turn(STEP_LONG);
  legFR.Turn(STEP_SHORT);
  legBL.Turn(STEP_LONG);
  legBR.Turn(STEP_SHORT);
}

void legUp(){
  legFL.LegUp();
  legFR.LegUp();
  legBL.LegUp();
  legBR.LegUp();
}

void legDown(){
  legFL.LegDown();
  legFR.LegDown();
  legBL.LegDown();
  legBR.LegDown();
}

void stepUp(){
  legFL.StepUp();
  legFR.StepUp();
  legBL.StepUp();
  legBR.StepUp();
}
```

```
void stepDown(){
  legFL.StepDown();
  legFR.StepDown();
  legBL.StepDown();
  legBR.StepDown();
}

void changeTrajectory() {
  legFL.ChangeTrajectory();
  legFR.ChangeTrajectory();
  legBL.ChangeTrajectory();
  legBR.ChangeTrajectory();
}

void updateState() {
  if (STATE != HOME && ctrlData.home) {
    STATE = HOME;
    Console.println("HOME");
  } else if (STATE != FORWARD && ctrlData.forward) {
    STATE = FORWARD;
    Console.println("FORWARD");
  } else if (STATE != REVERSE && ctrlData.backward) {
    STATE = REVERSE;
    Console.println("REVERSE");
  } else if (STATE != LEFT && ctrlData.left) {
    STATE = LEFT;
    Console.println("LEFT");
  } else if (STATE != RIGHT && ctrlData.right) {
    STATE = RIGHT;
    Console.println("RIGHT");
  } else if (STATE != LEG_UP && ctrlData.legUp) {
    STATE = LEG_UP;
    Console.println("LEG_UP");
  } else if (STATE != LEG_DOWN && ctrlData.legDown) {
    STATE = LEG_DOWN;
    Console.println("LEG_DOWN");
  } else if (STATE != STEP_UP && ctrlData.stepUp) {
    STATE = STEP_UP;
    Console.println("STEP_UP");
  } else if (STATE != STEP_DOWN && ctrlData.stepDown) {
    STATE = STEP_DOWN;
    Console.println("STEP_DOWN");
  } else if (STATE != CHANGE_TRAJECTORY && ctrlData.changeTrajectory) {
    STATE = CHANGE_TRAJECTORY;
    Console.println("CHANGE_TRAJECTORY");
  } else if (STATE != STOPPED && ctrlData.stopped) {
    STATE = STOPPED;
    Console.println("STOPPED");
  } else if (STATE != RESTART && ctrlData.restart) {
```

```
    STATE = RESTART;
    Console.println("RESTART");
  }
}


void setup() {
  Wire.begin();
  delay(10);

  batterymonitor.begin();
  delay(10);

  imu.begin();
  delay(10);

  servodriver.begin();
  delay(10);

  blecontroller.begin();
  delay(10);
}

void loop() {
  if (blecontroller.isConnected()) {
    blecontroller.getData(ctrlData);
  }

  if (batterymonitor.isInitialised()) {
    batterymonitor.getData(batteryData);
  }

  if (imu.isInitialised()) {
    imu.getData(imuData);
  }

  updateState();

  switch (STATE) {
    case HOME:
      home();
      break;
    case FORWARD:
      forward();
      break;
    case REVERSE:
      reverse();
      break;
    case LEFT:
```

```
      left();
      break;
    case RIGHT:
      right();
      break;
    case LEG_UP:
      legUp();
      break;
    case LEG_DOWN:
      legDown();
      break;
    case STEP_UP:
      stepUp();
      break;
    case STEP_DOWN:
      stepDown();
      break;
    case CHANGE_TRAJECTORY:
      changeTrajectory();
      break;
    case STOPPED:
      break;
    case RESTART:
      ESP.restart();
      break;
  }

  delay(10);
}
```

Servodriver.h

```
#ifndef SERVO_DRIVER_H
#define SERVO_DRIVER_H

#include <ArduinoConsole.h>
#include <Adafruit_PWMServoDriver.h>
#include <Wire.h>
#include "constants.h"

class ServoDriver {
public:

    static constexpr uint16_t SERVO_MAX_PULSE[SERVO_NUMBER] = {544,
548, 548, 544, 544, 548, 544, 548};
    static constexpr uint16_t SERVO_MIN_PULSE[SERVO_NUMBER] = {108,
108, 108, 108, 108, 108, 108, 108};
```

```
    static constexpr uint16_t SERVO_TOP[SERVO_NUMBER] = {
        SERVO_MIN_PULSE[0],
        SERVO_MAX_PULSE[1],
        SERVO_MAX_PULSE[2],
        SERVO_MIN_PULSE[3],
        SERVO_MIN_PULSE[4],
        SERVO_MAX_PULSE[5],
        SERVO_MIN_PULSE[6],
        SERVO_MAX_PULSE[7]
    };
    static constexpr uint16_t SERVO_BOTTOM[SERVO_NUMBER] = {
        SERVO_MAX_PULSE[0],
        SERVO_MIN_PULSE[1],
        SERVO_MIN_PULSE[2],
        SERVO_MAX_PULSE[3],
        SERVO_MAX_PULSE[4],
        SERVO_MIN_PULSE[5],
        SERVO_MAX_PULSE[6],
        SERVO_MIN_PULSE[7]
    };

    ServoDriver();
    void begin();
    void driveServo(uint8_t servo, double angle);
    bool isInitialised();

private:
    Adafruit_PWMServoDriver  _servodriver;
    long pulseFromAngle(uint8_t servo, double angle);
    bool _initialised;
};

#endif
```

Servodriver.cpp

```
#include "servodriver.h"

constexpr uint16_t ServoDriver::SERVO_MAX_PULSE[SERVO_NUMBER];
constexpr uint16_t ServoDriver::SERVO_MIN_PULSE[SERVO_NUMBER];
constexpr uint16_t ServoDriver::SERVO_TOP[SERVO_NUMBER];
constexpr uint16_t ServoDriver::SERVO_BOTTOM[SERVO_NUMBER];

ServoDriver::ServoDriver() : _servodriver(Adafruit_PWMServoDriver()),
_initialised(false) {}
```

```
void ServoDriver::begin(){
    Console.println("Initialise Adafruit PCA9685");

    if (!_servodriver.begin()) {
        Console.println("Failed to find PCA9685 chip");
        _initialised = false;
        return;
    }

    _servodriver.reset();
    _servodriver.setOutputMode(true);
    _servodriver.setOscillatorFrequency(OSCILLATOR_FREQUENCY);
    _servodriver.setPWMFreq(SERVO_FREQUENCY);

    _initialised = true;
}

bool ServoDriver::isInitialised() {
    return _initialised;
}

void ServoDriver::driveServo(uint8_t servo, double angle) {
    long pulse = pulseFromAngle(servo, angle);

    if (pulse < 0) {
        return;
    }

    _servodriver.setPin(servo, pulse);
}

long ServoDriver::pulseFromAngle(uint8_t servo, double angle) {
    if (servo < 0 || servo >= SERVO_NUMBER || angle < 0 || angle >
SERVO_MAX_ANGLE) {
        return -1;
    }

    return map(angle, SERVO_MIN_ANGLE, SERVO_MAX_ANGLE,
SERVO_BOTTOM[servo], SERVO_TOP[servo]);
}
```

Leg.h

```
#ifndef LEG_H
#define LEG_H
```

```
#include <ArduinoConsole.h>
#include "servodriver.h"
#include "constants.h"

class Leg {
public:
    struct KinematicsData{
        double x;
        double y;
        double phi_2;
        double phi_5;
    };

    struct TrajectoryData{
        KinematicsData kinematics[RESOLUTION*2];
        int size;
    };

    Leg(ServoDriver& servodriver, int servo_phi_2, int servo_phi_5,
bool isFront, int gaitOffset);
    void driveFullStep(double stepLength);
    void driveNextStep(double stepLength);
    void Home();
    void Reverse();
    void Forward();
    void Turn(double stepLength);
    void LegUp();
    void LegDown();
    void StepUp();
    void StepDown();
    void ChangeTrajectory();

private:
    // Servo
    ServoDriver& servodriver;
    int servo_phi_2;
    int servo_phi_5;
    void driveLeg(double phi_2, double phi_5);

    // Parameter Change
    void updateStepOffset();
    void updateStepHeight();

    // Trajectory
    int stepindex;
    bool isCycloidal;
    bool isFront;
    bool isReverse;
    int gaitOffset;
```

```
    int stepOffset;
    int stepHeight;
    void generateFullStep(TrajectoryData& data, double stepLength);
    void generateNextStep(KinematicsData& data, double stepLength, int
idx);
    void cycloidalTrajectory(KinematicsData& data, double stepLength,
int idx);
    void sinusoidalTrajectory(KinematicsData& data, double stepLength,
int idx);

    // Kinematics
    void ik(KinematicsData& data, double x, double y);
    void fk(KinematicsData& data, double phi_2, double phi_5);
    double mod(double a, double b);
};

#endif
```

Leg.cpp

```
#include "leg.h"

Leg::Leg(ServoDriver& servodriver, int servo_phi_2, int servo_phi_5,
bool isFront, int gaitOffset) :
    servodriver(servodriver),
    servo_phi_2(servo_phi_2),
    servo_phi_5(servo_phi_5),
    isFront(isFront),
    gaitOffset(gaitOffset),
    stepOffset(WORKSPACE_Y_MIN),
    stepHeight(STEP_HEIGHT_MIN),
    isReverse(false),
    isCycloidal(true),
    stepindex(0 + gaitOffset) {
}

void Leg::Home() {
    KinematicsData data;
    stepindex = 0 + gaitOffset;
    driveNextStep(100);
    stepindex = 0 + gaitOffset;
}

void Leg::driveLeg(double phi_2, double phi_5) {
    servodriver.driveServo(servo_phi_2, phi_2);
    servodriver.driveServo(servo_phi_5, phi_5);
}
```

```
void Leg::driveFullStep(double stepLength) {
    TrajectoryData data;
    generateFullStep(data, stepLength);
    for (int i = 0; i < data.size; i++) {
        driveLeg(data.kinematics[i].phi_2, data.kinematics[i].phi_5);
        delay(15);
    }
    stepindex = 0 + gaitOffset;
}

void Leg::driveNextStep(double stepLength) {
    KinematicsData data;
    generateNextStep(data, stepLength, stepindex);
    driveLeg(data.phi_2, data.phi_5);
    stepindex = isReverse ? (stepindex - 1) : (stepindex + 1);
    if (stepindex >= RESOLUTION) stepindex = 0;
    if (stepindex < 0) stepindex = RESOLUTION - 1;
}

void Leg::generateFullStep(TrajectoryData& data, double stepLength) {
    data.size = RESOLUTION;

    for (int i = 0; i < RESOLUTION; i++) {
        generateNextStep(data.kinematics[i], stepLength, i);
    }
}

void Leg::generateNextStep(KinematicsData& data, double stepLength, int
idx) {
    if (isCycloidal) {
        cycloidalTrajectory(data, stepLength, idx);
    } else {
        sinusoidalTrajectory(data, stepLength, idx);
    }
}

void Leg::sinusoidalTrajectory(KinematicsData& data, double stepLength,
int idx) {
    if (idx < LAMBDA * RESOLUTION){
        double sigma = (RAD_360 * idx) / ((LAMBDA * RESOLUTION) - 1);
        double x = stepLength * sigma / RAD_360 - (stepLength/2);
        ik(data,
            x,
            -(stepHeight * sin((x * PI / stepLength) + RAD_90)) +
(stepOffset + stepHeight)
        );
    }
    else if (idx < RESOLUTION){
        double dx = stepLength / (((1-LAMBDA) * RESOLUTION) - 1);
```

```
        ik(data,
            (stepLength/2) - (idx-(LAMBDA * RESOLUTION)) * dx,
            stepOffset + stepHeight
        );
    }
}

void Leg::cycloidalTrajectory(KinematicsData& data, double stepLength,
int idx) {
    if (idx < LAMBDA * RESOLUTION){
        double sigma = (RAD_360 * idx) / ((LAMBDA * RESOLUTION) - 1);
        ik(data,
            ((stepLength * (sigma - sin(sigma)) / (RAD_360))) + (-
stepLength/2) ,
            -((stepHeight * (1 - cos(sigma)) / 2)) + (stepOffset +
stepHeight)
        );
    }
    else if (idx < RESOLUTION){
        double dx = stepLength / (((1-LAMBDA) * RESOLUTION) - 1);
        ik(data,
            (stepLength/2) - (idx-(LAMBDA * RESOLUTION)) * dx,
            stepOffset + stepHeight
        );
    }
}

void Leg::ik(KinematicsData& data, double x, double y) {
    data.x = x;
    data.y = y;

    if(x < WORKSPACE_X_MIN || x > WORKSPACE_X_MAX || y <
WORKSPACE_Y_MIN || y > WORKSPACE_Y_MAX) {
        Console.printf("IK out of bounds: x=%.2f, y=%.2f\n", x, y);
        data.phi_2 = -1;
        data.phi_5 = -1;
        return;
    }

    // Calculate coefficients
    double A_2 = -L_2_5 * (2*x + L_1);
    double A_5 = -L_2_5 * (2*x - L_1);
    double B_2_5 = -2 * L_2_5 * y;
    double C_2 = x*x + y*y + L_1*L_1/ 4 + L_2_5*L_2_5 - L_3_4*L_3_4 +
L_1*x;
    double C_5 = C_2 - 2*L_1*x;

    // Calculate angles
    double phi_2_sqrt = sqrt(A_2*A_2 + B_2_5*B_2_5 - C_2*C_2);
```

```
    double phi_5_sqrt = sqrt(A_5*A_5 + B_2_5*B_2_5 - C_5*C_5);

    double phi_2_n = mod(2.0 * atan2((B_2_5 - phi_2_sqrt), (A_2 -
C_2)), RAD_360);
    double phi_2_p = mod(2.0 * atan2((B_2_5 + phi_2_sqrt), (A_2 -
C_2)), RAD_360);

    double phi_5_n = mod(2.0 * atan2((B_2_5 - phi_5_sqrt), (A_5 -
C_5)), RAD_360);
    double phi_5_p = mod(2.0 * atan2((B_2_5 + phi_5_sqrt), (A_5 -
C_5)), RAD_360);

    // Select correct angle
    data.phi_2 = (phi_2_n >= RAD_90 && phi_2_n <= RAD_270) ? phi_2_n :
phi_2_p;
    data.phi_5 = ((phi_5_n >= 0.0 && phi_5_n <= RAD_90) || (phi_5_n >=
RAD_270 && phi_5_n <= RAD_360)) ? phi_5_n : phi_5_p;

    data.phi_2 = (data.phi_2 - RAD_90)*RAD_TO_DEG;
    data.phi_5 = (data.phi_5 >= RAD_270 ? RAD_450 - data.phi_5 : RAD_90
- data.phi_5)*RAD_TO_DEG;

    return;
}

void Leg::fk(KinematicsData& data, double phi_2, double phi_5) {
    phi_2 = phi_2*DEG_TO_RAD;
    phi_2 += RAD_90;
    phi_5 = phi_5*DEG_TO_RAD;
    phi_5 = phi_5 >= RAD_270 ? RAD_450 - phi_5 : RAD_90 - phi_5;

    double L_s_x = L_1 + L_2_5*cos(phi_5) - L_2_5*cos(phi_2);
    double L_s_y = L_2_5*cos(phi_5) - L_2_5*cos(phi_2);
    double L_s = sqrt(L_s_x*L_s_x + L_s_y*L_s_y);
    double phi_s = atan2(L_s_y, L_s_x)+PI;

    double phi = acos(-L_s*L_s / (2*L_s*L_3_4));

    double phi_3 = phi_s - phi;
    double phi_4 = phi_s + phi;

    data.x = -L_1/2 + L_2_5*cos(phi_2) + L_3_4*cos(phi_3);
    data.y = L_2_5*sin(phi_2) + L_3_4*sin(phi_3);

    return;
}

double Leg::mod(double a, double b) {
    return a - b * floor(a / b);
```

```
}

void Leg::Reverse() {
    isReverse = true;
    driveNextStep(STEP_LONG);
}

void Leg::Forward() {
    isReverse = false;
    driveNextStep(STEP_LONG);
}

void Leg::Turn(double stepLength) {
    isReverse = false;
    driveNextStep(stepLength);
}

void Leg::updateStepOffset() {
    if (stepOffset + stepHeight > WORKSPACE_Y_MAX) {
        stepOffset = WORKSPACE_Y_MAX - stepHeight;
    }
    if (stepOffset < WORKSPACE_Y_MIN) {
        stepOffset = WORKSPACE_Y_MIN;
    }
}

void Leg::updateStepHeight() {
    if (stepHeight > STEP_HEIGHT_MAX) {
        stepHeight = STEP_HEIGHT_MAX;
    }
    if (stepHeight < STEP_HEIGHT_MIN) {
        stepHeight = STEP_HEIGHT_MIN;
    }
}

void Leg::LegUp() {
    stepOffset++;

    updateStepOffset();

    stepindex = isReverse ? (stepindex + 1) : (stepindex - 1);
    if (stepindex >= RESOLUTION) stepindex = 0;
    if (stepindex < 0) stepindex = RESOLUTION - 1;

    driveNextStep(STEP_LONG);
}

void Leg::LegDown() {
    stepOffset--;
```

```
    updateStepOffset();

    stepindex = isReverse ? (stepindex + 1) : (stepindex - 1);
    if (stepindex >= RESOLUTION) stepindex = 0;
    if (stepindex < 0) stepindex = RESOLUTION - 1;

    driveNextStep(STEP_LONG);
}

void Leg::StepUp() {
    stepHeight++;

    updateStepHeight();
    updateStepOffset();

    stepindex = isReverse ? (stepindex + 1) : (stepindex - 1);
    if (stepindex >= RESOLUTION) stepindex = 0;
    if (stepindex < 0) stepindex = RESOLUTION - 1;

    driveNextStep(STEP_LONG);
}

void Leg::StepDown() {
    stepHeight--;

    updateStepHeight();
    updateStepOffset();

    stepindex = isReverse ? (stepindex + 1) : (stepindex - 1);
    if (stepindex >= RESOLUTION) stepindex = 0;
    if (stepindex < 0) stepindex = RESOLUTION - 1;

    driveNextStep(STEP_LONG);
}

void Leg::ChangeTrajectory() {
    isCycloidal = !isCycloidal;
}
```

Imu.h

```
#ifndef IMU_H
#define IMU_H

#include <ArduinoConsole.h>
#include <Adafruit_ICM20X.h>
#include <Adafruit_ICM20948.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include "constants.h"

class IMU {
public:
    struct IMUData {
        sensors_event_t accelerometer;
        sensors_event_t gyroscope;
        sensors_event_t magnetometer;
        sensors_event_t temperature;
    };

    IMU();
    void begin();
    void printData(IMUData& data);
    void getData(IMUData& data);
    bool isInitialised();

private:
    Adafruit_ICM20948 _imu;
    bool _initialised;
};


#endif
```

Imu.cpp

```
#include "imu.h"

IMU::IMU() : _imu(Adafruit_ICM20948()), _initialised(false) {}

void IMU::begin() {
    Console.println("Initialise Adafruit ICM20948");

    if (!_imu.begin_I2C()) {
        Console.println("Failed to find ICM20948 chip");
        _initialised = false;
        return;
```

```
    }

    Console.println("Found ICM20948 chip");

    _imu.setAccelRange(ICM20948_ACCEL_RANGE_4_G);
    _imu.setGyroRange(ICM20948_GYRO_RANGE_500_DPS);
    _imu.setAccelRateDivisor(IMU_ACCELEROMETER_DIVISOR);
    _imu.setGyroRateDivisor(IMU_GYROSCOPE_DIVISOR);
    _imu.setMagDataRate(AK09916_MAG_DATARATE_10_HZ);

    _initialised = true;
}

bool IMU::isInitialised() {
    return _initialised;
}

void IMU::getData(IMUData& data) {
    _imu.getEvent(&data.accelerometer, &data.gyroscope,
&data.temperature, &data.magnetometer);
}

void IMU::printData(IMUData& data) {
    Console.printf("\nTemp: %d *C  ", data.temperature.temperature);
    Console.printf("Acc X: %d Y: %d Z: %d m/s^2  ",
data.accelerometer.acceleration.x, data.accelerometer.acceleration.y,
data.accelerometer.acceleration.z);
    Console.printf("Mag X: %d Y: %d Z: %d uT  ",
data.magnetometer.magnetic.x, data.magnetometer.magnetic.y,
data.magnetometer.magnetic.z);
    Console.printf("Gyro X: %d Y: %d Z: %d rad/s  ",
data.gyroscope.gyro.x, data.gyroscope.gyro.y, data.gyroscope.gyro.z);
}
```

Controller.h

```
#ifndef CONTROLLER_H
#define CONTROLLER_H

#include "sdkconfig.h"
#include <Bluepad32.h>

class BLEController {
public:
    struct BLEControllerData {
        bool home;
        bool restart;
        bool left;
```

```
        bool right;
        bool forward;
        bool backward;
        bool stopped;
        bool legUp;
        bool legDown;
        bool stepUp;
        bool stepDown;
        bool changeTrajectory;
        uint32_t leftRight;
        uint32_t upDown;
    };

    BLEController();

    void begin();
    void printData(BLEControllerData& data);
    void getData(BLEControllerData& data);
    bool isConnected();


private:
    ControllerPtr ctrl;
    void _onConnected(ControllerPtr ctl);
    void _onDisconnected(ControllerPtr ctl);

    static void onConnected(ControllerPtr ctl);
    static void onDisconnected(ControllerPtr ctl);

    static BLEController* instance;
};

#endif
```

Controller.cpp

```
#include "controller.h"

BLEController* BLEController::instance = nullptr;

BLEController::BLEController() : ctrl(nullptr) { instance = this; }

void BLEController::begin() {
    Console.printf("Firmware: %s\n", BP32.firmwareVersion());
    const uint8_t* addr = BP32.localBdAddress();
    Console.printf("BD Addr: %2X:%2X:%2X:%2X:%2X:%2X\n", addr[0],
addr[1], addr[2], addr[3], addr[4], addr[5]);
```

```
    BP32.setup(&onConnected, &onDisconnected, true);
    BP32.enableBLEService(false);
}

void BLEController::onConnected(ControllerPtr ctl) {
    if (instance) instance->_onConnected(ctl);  // forward to the real
instance
}

void BLEController::_onConnected(ControllerPtr ctl) {
    bool foundEmptySlot = false;

    if (ctrl == nullptr) {
        Console.print("CALLBACK: Controller is connected\n");
        ControllerProperties properties = ctl->getProperties();
        Console.printf("Controller model: %s, VID=0x%04x,
PID=0x%04x\n", ctl->getModelName(), properties.vendor_id,
                       properties.product_id);
        ctrl = ctl;
        foundEmptySlot = true;
    }

    if (!foundEmptySlot) {
        Console.println("CALLBACK: Controller connected, but could not
find empty slot");
    }
}

void BLEController::onDisconnected(ControllerPtr ctl) {
    if (instance) instance->_onDisconnected(ctl);  // forward to the
real instance
}

void BLEController::_onDisconnected(ControllerPtr ctl) {
    bool foundController = false;

    if (ctrl == ctl) {
        Console.print("CALLBACK: Controller disconnected\n");
        ctrl = nullptr;
        foundController = true;
    }

    if (!foundController) {
        Console.println("CALLBACK: Controller disconnected, but not
found");
    }
}

bool BLEController::isConnected() {
```

```
    return (ctrl != nullptr) && ctrl->isConnected();
}

void BLEController::printData(BLEControllerData& data) {
    if (ctrl == nullptr) {
        return;
    }

    Console.printf(
        "idx: %d, home: %d, restart: %d, left: %d, right: %d, forward:
%d, backward: %d, legUp: %d, legDown: %d, stepUp: %d, stepDown: %d,
changeTrajectory: %d, stopped: %d, x-axis: %4d, y-axis: %4d\n",
        data.home,
        data.restart,
        data.left,
        data.right,
        data.forward,
        data.backward,
        data.legUp,
        data.legDown,
        data.stepUp,
        data.stepDown,
        data.changeTrajectory,
        data.stopped,
        data.leftRight,
        data.upDown
    );
}

void BLEController::getData(BLEControllerData& data) {
    if (!BP32.update()){
        return;
    }

    if (ctrl && ctrl->isConnected() && ctrl->hasData()) {
        if (ctrl->isGamepad()) {
            data.home = ctrl->miscButtons() == 0x01;
            data.restart = ctrl->miscButtons() == 0x08;
            data.left = ctrl->dpad() == 0x08;
            data.right = ctrl->dpad() == 0x04;
            data.forward = ctrl->dpad() == 0x01;
            data.backward = ctrl->dpad() == 0x02;
            data.legUp = ctrl->throttle() >= 512;
            data.legDown = ctrl->brake() >= 512;
            data.stepUp = ctrl->buttons() == 0x0010;
            data.stepDown = ctrl->buttons() == 0x0020;
            data.changeTrajectory = ctrl->buttons() == 0x0008;
            data.stopped = !data.forward && !data.backward &&
!data.left && !data.right && !data.home && !data.restart && !data.legUp
```

```
&& !data.legDown && !data.stepUp && !data.stepDown &&
!data.changeTrajectory;
            data.leftRight = ctrl->axisX();
            data.upDown = ctrl->axisY();
        }
        else if (ctrl->isKeyboard()) {
            data.home = ctrl->isKeyPressed(Keyboard_H);
            data.restart = ctrl->isKeyPressed(Keyboard_R);
            data.left = ctrl->isKeyPressed(Keyboard_LeftArrow) || ctrl-
>isKeyPressed(Keyboard_A);
            data.right = ctrl->isKeyPressed(Keyboard_RightArrow) ||
ctrl->isKeyPressed(Keyboard_D);
            data.forward = ctrl->isKeyPressed(Keyboard_UpArrow) ||
ctrl->isKeyPressed(Keyboard_W);
            data.backward = ctrl->isKeyPressed(Keyboard_DownArrow) ||
ctrl->isKeyPressed(Keyboard_S);
            data.legUp = ctrl->isKeyPressed(Keyboard_Q);
            data.legDown = ctrl->isKeyPressed(Keyboard_E);
            data.stepUp = ctrl->isKeyPressed(Keyboard_Z);
            data.stepDown = ctrl->isKeyPressed(Keyboard_X);
            data.changeTrajectory = ctrl->isKeyPressed(Keyboard_T);
            data.stopped = !data.forward && !data.backward &&
!data.left && !data.right && !data.home && !data.restart && !data.legUp
&& !data.legDown && !data.stepUp && !data.stepDown &&
!data.changeTrajectory;
            data.leftRight = 0;
            data.upDown = 0;
        }
    }
}
```

Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#include <Arduino.h>

#define INA228_SHUNT_RESISTANCE 0.015
#define INA228_MAX_CURRENT 10.0

#define IMU_ACCELEROMETER_DIVISOR 255
#define IMU_GYROSCOPE_DIVISOR 255

#define RAD_90 (PI/2)
#define RAD_270 (3*PI/2)
#define RAD_360 (2*PI)
#define RAD_450 (5*PI/2)
```

```
#define L_1 50.0
#define L_2_5 48.5
#define L_3_4 92.8

#define OSCILLATOR_FREQUENCY 25000000
#define SERVO_FREQUENCY 50
#define SERVO_NUMBER 8
#define SERVO_MAX_ANGLE 180.0
#define SERVO_MIN_ANGLE 0.0

#define SERVO_FLB 0 // 108, 544
#define SERVO_FLF 1 // 108, 548
#define SERVO_BLF 2 // 108, 544
#define SERVO_BLB 3 // 108, 548
#define SERVO_BRF 4 // 108, 544
#define SERVO_BRB 5 // 108, 548
#define SERVO_FRF 6 // 108, 544
#define SERVO_FRB 7 // 108, 548

#define RESOLUTION 100

#define WORKSPACE_X_MIN -50
#define WORKSPACE_X_MAX 50
#define WORKSPACE_Y_MIN 50
#define WORKSPACE_Y_MAX 100

#define STEP_HEIGHT_MIN 20
#define STEP_HEIGHT_MAX 50
#define STEP_LONG 100
#define STEP_SHORT 30

#define LAMBDA 0.25

#define HOME 0
#define FORWARD 1
#define REVERSE 2
#define LEFT 3
#define RIGHT 4
#define LEG_UP 5
#define LEG_DOWN 6
#define STEP_UP 7
#define STEP_DOWN 8
#define RESTART 9
#define STOPPED 10
#define CHANGE_TRAJECTORY 11

#endif
```

Batterymonitor.h

```
#ifndef BATTERY_MONITOR_H
#define BATTERY_MONITOR_H

#include <ArduinoConsole.h>
#include <Adafruit_INA228.h>
#include "constants.h"

class BatteryMonitor {
public:
    struct BatteryData {
        float current;
        float voltage;
        float shuntvoltage;
        float power;
        float energy;
        float charge;
        float temperature;
    };

    BatteryMonitor();
    void begin();
    void printData(BatteryData& data);
    void getData(BatteryData& data);
    bool isInitialised();

private:
    Adafruit_INA228 _batterymonitor;
    bool _initialised;
};

#endif
```

Batterymonitor.cpp

```
#include "batterymonitor.h"

BatteryMonitor::BatteryMonitor() : _batterymonitor(Adafruit_INA228()),
_initialised(false) {}

void BatteryMonitor::begin() {
    Console.println("Initialise Adafruit INA228");

    if (!_batterymonitor.begin()) {
        Console.println("Failed to find INA228 chip");
        _initialised = false;
        return;
    }

    Console.println("Found INA228 chip\n");
    _batterymonitor.setShunt(INA228_SHUNT_RESISTANCE,
INA228_MAX_CURRENT);
    _batterymonitor.setAveragingCount(INA228_COUNT_16);
    _batterymonitor.setVoltageConversionTime(INA228_TIME_150_us);
    _batterymonitor.setCurrentConversionTime(INA228_TIME_280_us);
    _initialised = true;
}

bool BatteryMonitor::isInitialised() {
    return _initialised;
}

void BatteryMonitor::getData(BatteryData& data) {
    data.current = _batterymonitor.getCurrent_mA();
    data.voltage = _batterymonitor.getBusVoltage_V();
    data.shuntvoltage = _batterymonitor.getShuntVoltage_mV();
    data.power = _batterymonitor.getPower_mW();
    data.energy = _batterymonitor.readEnergy();
    data.charge = _batterymonitor.readCharge();
    data.temperature = _batterymonitor.readDieTemp();
}

void BatteryMonitor::printData(BatteryData& data) {
    Console.printf("\nA: %d mA  ", data.current);
    Console.printf("V: %d V  ", data.voltage);
    Console.printf("SV: %d mV  ", data.shuntvoltage);
    Console.printf("P: %d mW  ", data.power);
    Console.printf("E: %d J  ", data.energy);
    Console.printf("C: %d C  ", data.charge);
    Console.printf("Temp: %d *C  ", data.temperature);
}
```

Appendix F: Power Consumption Testing Results

The following graphs depict the values of the power characteristics recorded by the onboard battery monitor over time.

Forward Motion

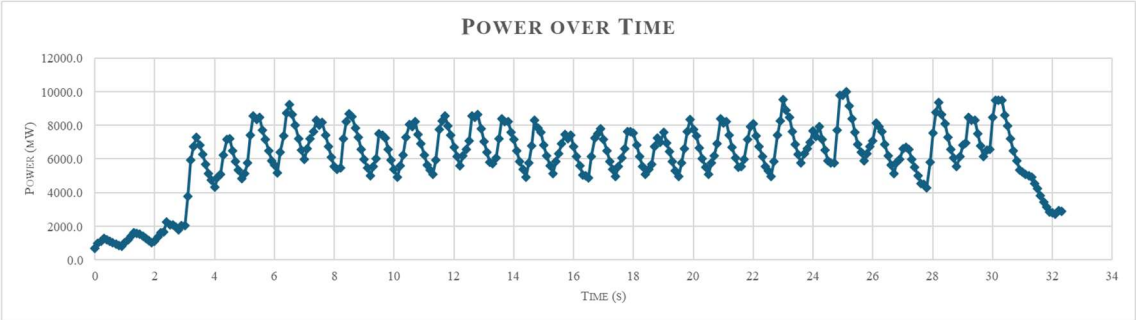

Figure 85: Power Readings During a Forward Motion Over Time.

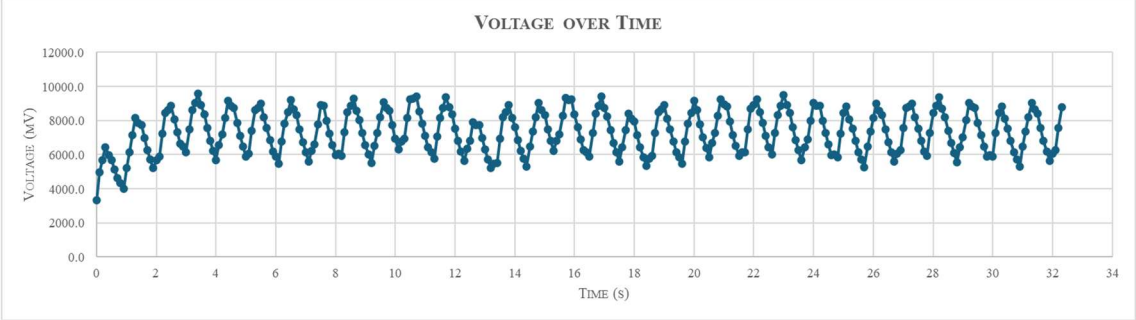

Figure 86: Voltage Readings During a Forward Motion Over Time.

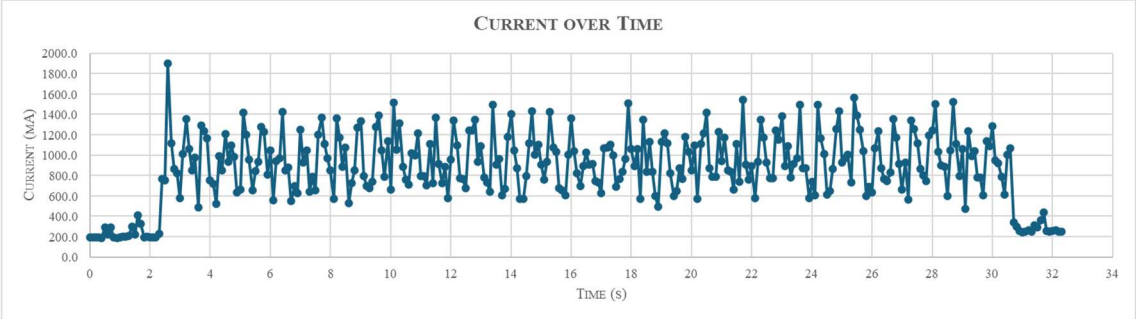

Figure 87: Current Readings During a Forward Motion Over Time.

Reverse Motion

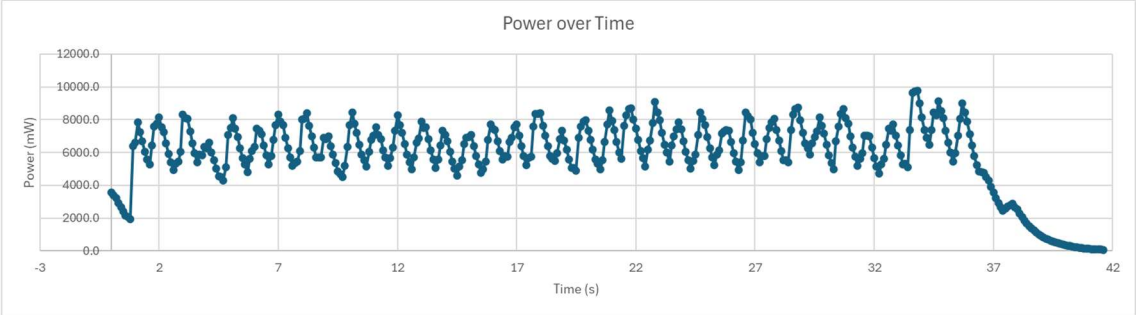

Figure 88: Power Readings During a Reverse Motion Over Time.

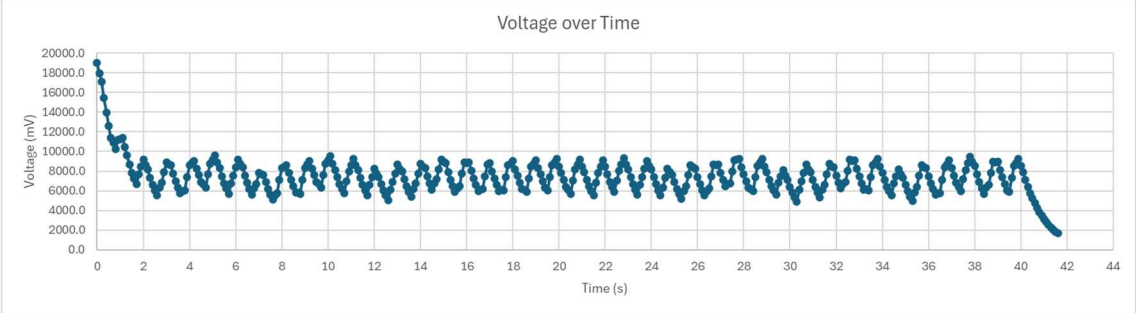

Figure 89: Voltage Readings During a Reverse Motion Over Time.

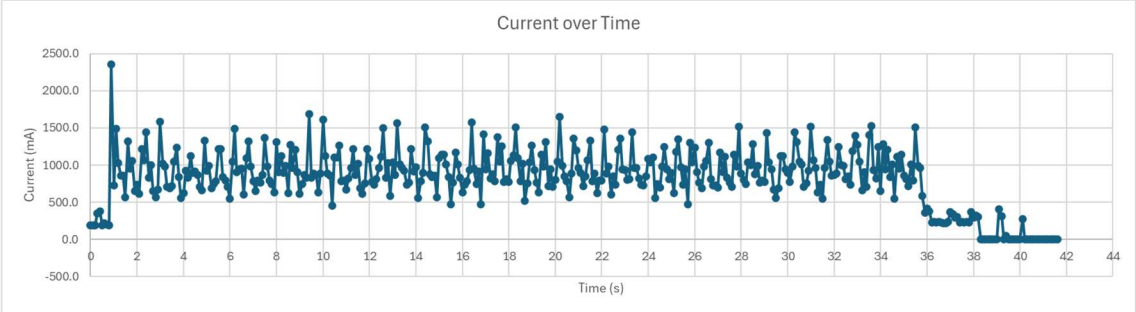

Figure 90: Current Readings During a Reverse Motion Over Time.

Left Motion

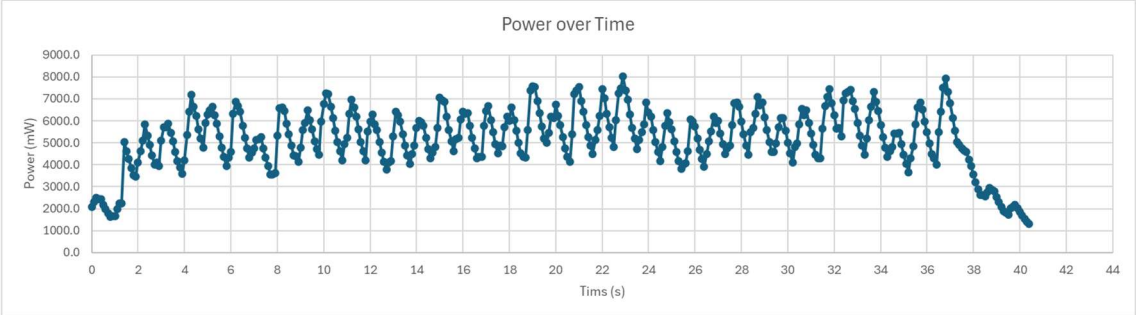

Figure 91: Power Readings During a Left Motion Over Time.

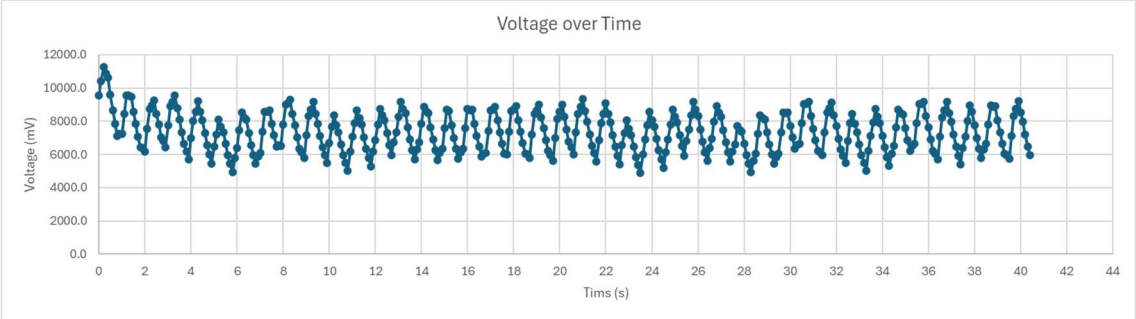

Figure 92: Voltage Readings During a Left Motion Over Time.

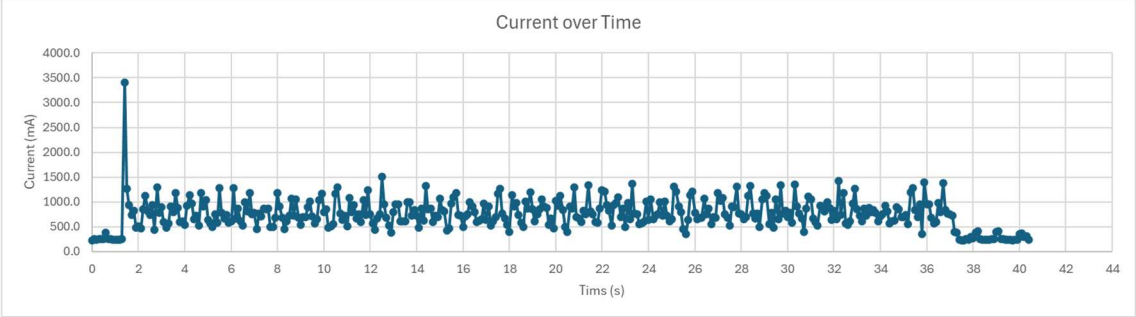

Figure 93: Current Readings During a Left Motion Over Time.

Right Motion

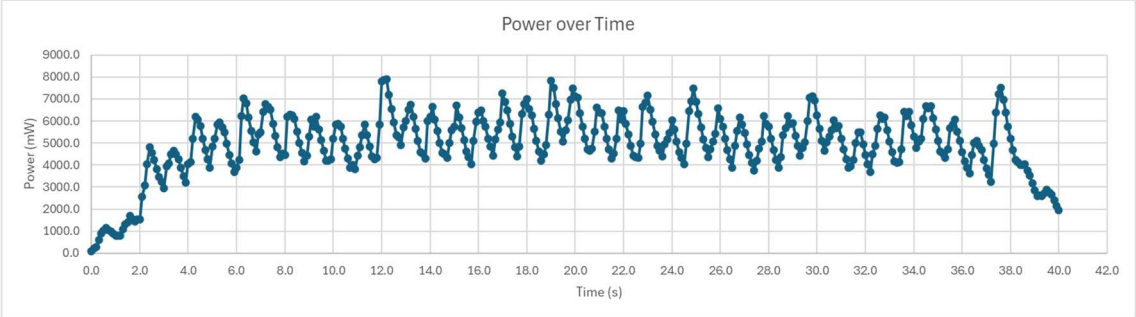

Figure 94: Power Readings During a Right Motion Over Time.

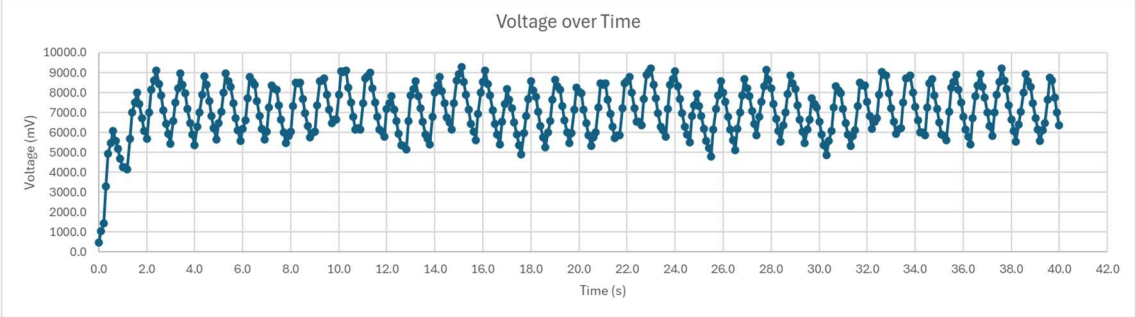

Figure 95: Voltage Readings During a Right Motion Over Time.

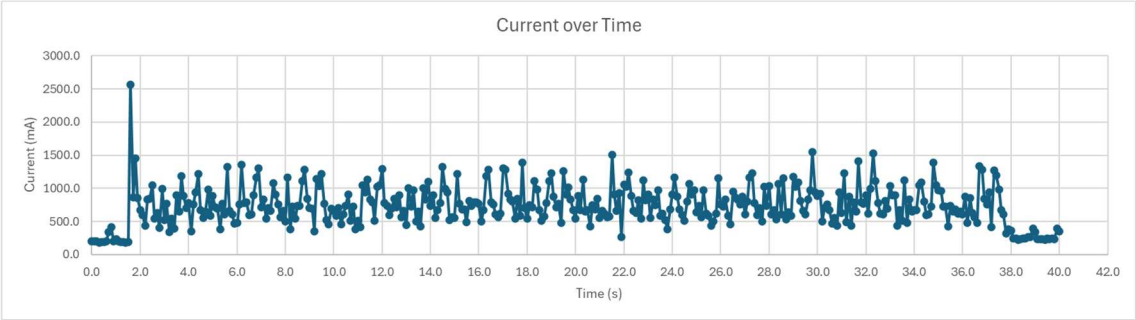

Figure 96: Current Readings During a Right Motion Over Time.