Mathematical Modelling and Control System Development of a Remote Controlled, IMU Stabilised Hexapod Robot

Presented by:
Ross Christopher
Supervised by Associate Professor Hendrik D. Mouton

Robotics and Agents Research Laboratory
University of Cape Town

Submitted to the Department of Mechanical Engineering at the University of Cape Town in fulfilment of the academic requirements for a Master of Science in Engineering.

November 2019

Plagiarism Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as his or her own work.
5. I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis/dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed: Signed by candidate Date: May 1, 2020

Ross Christopher

Abstract

Walking robots are useful in search and rescue applications due to their ability to navigate uneven and complex terrain.

A hexapod robot has been developed by the Robotics and Agents Research Lab at UCT, however multiple inadequacies have become evident. This work aims to produce a mathematical model of the hexapod and using this model, implement an effective control algorithm to achieve a smooth walking motion and overcome the original flaws.

The mathematical model was integrated with the mechanical structure of the hexapod and controlled by a micro-controller. This micro-controller allows for a rapid start-up and low power consumption when compared to previous iterations of the hexapod. Using a path generation algorithm sets of foot positions and velocities are generated. Generating these points in real time allows for walking in any direction without any pre-defined foot positions.

To enable attitude control of the hexapod body, an inertial measurement unit was added to the hexapod. By using a PID controller the IMU pitch and roll data was used to control a height offset of each foot of the hexapod, allowing for stabilisation of the hexapod body.

An improved wireless remote control was developed to facilitate communication with a host computer. The remote system has a graphical user interface allowing for walking control and status information feedback, such as error information and current battery voltage.

Walking tests have shown that the hexapod walks successfully with a smooth tripod gait using the path generation algorithm. Stabilisation tests have shown that the hexapod is capable of stabilising itself after a disturbance to its pitch and/or roll in ± 2.5 seconds with a steady state error of ± 0.001 radians.

This proves that the hexapod robot can be controlled wirelessly while walking in any direction with a stabilised body. This is beneficial in search and rescue as the hexapod has a high degree of manoeuvrability to access areas too dangerous for rescuers to access. With cameras mounted on the stabilised body, it can be used to locate survivors in a disaster area and assist rescuers in recovering them with speed.

Acknowledgements

Firstly I would like to thank the University of Cape Town for providing funding towards my Masters degree.

I would like to thank my supervisor Hennie Mouton who, over the years, has guided me through this project with enthusiasm and encouragement. Thank you for funding the project components, without which the hexapod could not have been completed.

To my parents Fred and Una and my brother Jimmy. Thank you for supporting me throughout my undergraduate and postgraduate degrees. Your input during this project, and all other personal projects, has been invaluable. Without your support I would not be where I am today.

Thanks to Sven Weihe for working with me on the hexapod; designing a battery and wireless charging system. Without your contribution the hexapod would never have been truly complete.

Finally, thanks to those who have worked with me in RARL, James Hepworth and John Ogundiran. Your input and friendship has helped me through this project every step of the way.

Table of Contents

List of Figures

List of Tables

Glossary

Anthropomorphism	”The attribution of human traits...to non-human entities” [1].
Attitude	The orientation of an object in space (pitch, roll and yaw).
End Effector	The manipulator at the end of the last link of a serial chain of links (i.e. a foot or gripper).
Pitch	Rotation around the x-axis.
Roll	Rotation around the y-axis.
Yaw	Rotation around the z-axis.

1 Introduction

Robots are becoming more and more useful each day with some of their aims including simplifying and automating production processes and aiding in every day tasks. In the past wheeled robots have been most common due to the simplicity of their design and the control system required to operate them. However they perform poorly on uneven terrain. Legged robots are developed to address this issue. Legged robots have a superior agility when compared to wheeled robots and allow for navigation over complex and uneven terrain. A four legged robot is common, however having more legs allows for the failure of one or more legs while the robot retains the ability to walk, albeit in a crippled state. In the case of a hexapod, a six legged robot, it has an increased agility which allows it to traverse uneven terrain and climb over objects [2]. Since it has more legs than needed for simple walking it allows for complex walking gaits and the failure of up to two legs while retaining the ability to walk. Because hexapods have redundant legs[1], aside from being able to function after losing functionality of some of its legs, it can re-purpose two of its legs into grippers which allows it to interact with objects while walking.

One such application of this is in search and rescue or disaster management. When a disaster, for example an earthquake, strikes it is crucial to get to the scene quickly and be able to move around safely while gathering information about possible survivors and the surroundings. Sending in a search and rescue team to do this is dangerous for the rescuers and could put more lives at risk. Sending in an agile robot with necessary sensors such as infrared cameras and global positioning systems (GPS) allows for search and rescue teams to identify further hazards in the surrounding area as well as locate any survivors. This allows for targeted rescue efforts which reduces the risk to the rescuers and will decrease the time needed to find and save people [3].

[1] Having more legs than required for basic motion

The work done in this project aims to continue the research and development of the hexapod robot in the University of Cape Town's Robotics and Agents Research Lab (RARL) by implementing an effective control system such that the robot can move with a high degree of agility and precision.

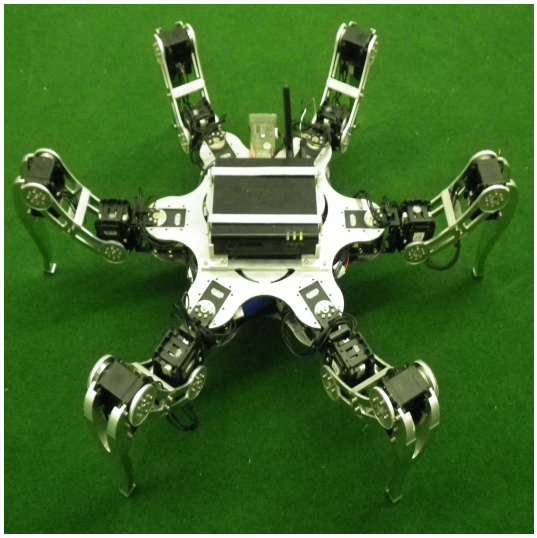


Figure 1.1: Hexapod Robot

1.1 Problem Statement

The University of Cape Town has developed a hexapod Robot with undergraduate students over several years [4], [5], [6]. At the start of 2018 it moved by reading in motor positions from a file line by line and moving each motor to its respective position. This shows no level of control and is simply a 'brute force' approach to robotic motion. It was controlled by an on-board computer running Windows 7 which led to a long start-up time (± 6 minutes). In search and rescue if the robot being utilised needs a significant amount of time to start-up then it is losing out on critical time during which it could be saving people.

This previous iteration of the hexapod is referred to as the 'MK1' hexapod from here on and its structure is discussed in section 3.

A control system needs to be developed in order to optimize the motion of the robot and allow for more flexible use. In order to achieve this a mathematical model must be developed to produce relationships between foot positions and motor angles in order to produce an effective control algorithm.

Once a control algorithm has been developed, robot applications such as search and rescue can be explored. The hexapod is an agile robot and allows for increased maneuverability in difficult terrain. A possible application includes mountain rescue. Placing a camera and tracking software aboard the hexapod will allow for the tracking and locating of persons who may be (for example) lost in a mountain. Note that this application would need a larger version of the hexapod to be able to climb over large rocks.

Research into robotic manipulators that mimic nature, or more specifically ones which mimic human motion, is beneficial in the field of prosthetics. The 3-DOF[2] anthropomorphic manipulators that make up the hexapod mimic the joints and motion of a human leg and as such the mathematics involved in this research could be translated to the field of bio-engineering and prosthetics design.

[2]Three degrees of freedom.

1.2 Aims and Objectives

The aim of this project is to gain a deeper understanding about robotics and robot control as well as learn the fundamental mathematics behind robotic systems and robotic motion. Various mathematical models of the hexapod's legs as well as the complete hexapod system is developed. This is done by determining the kinematics and dynamics of each system. A control system will be developed based on the mathematical models and refined until an effective control algorithm has been developed such that the hexapod can walk and move with a high degree of agility and maneuverability.

1.3 Scope and Limitations

This project includes the development of a mathematical model of the leg and body of the hexapod robot. The mathematical model includes the inverse and differential kinematics of a single leg. Body kinematics is used to translate this into a global coordinate system. The dynamics of a single leg, in the form of the equations of motion, is calculated to allow for the determination of the relationship between foot forces and motor torques.

The mathematical model is implemented on a micro-controller in C++.

A remote control system is developed to allow the hexapod to be controlled remotely from a host computer. This includes the development of a graphic user interface (GUI) in java for the host computer and relevant code on the hexapod to communicate with it.

An inertial measurement unit (IMU) will be added to the hexapod to allow for the determination of the hexapod's attitude[3]. A PID[4] controller will be developed so that, using the pitch and roll data from the IMU, the attitude of the hexapod body can be set and maintained over uneven terrain.

Various mechanical changes are made to the hexapod in order to improve its structure and performance.

Printed circuit boards (PCBs) are developed to include all circuitry necessary for the hexapod to operate, in a small form factor.

This project does not include the development of a battery and power regulation system for the hexapod, however during 2019 a battery and regulation system with wireless charging was developed for the hexapod by Sven Weihe, an undergraduate electro-mechanical engineer, during his final year project [7].

Although multiple gaits are possible with the mathematics and algorithms developed, only the tripod gait is implemented and tested. Gait selection and gait design is not a focus of this project and as such the fastest gait [8], the tripod gait, is the one of interest.

[3] The orientation of the hexapod (pitch, roll and yaw)
[4] Proportional-Integral-Derivative

1.4 Proposed Research Plan

The proposed research plan below was developed at the beginning of the project and as such is written in future tense.

The project outline includes producing a mathematical model of the hexapod robot. This has begun by modelling a single leg using inverse kinematics. Gaining the necessary knowledge to complete the modelling and producing a final model of one leg is expected to be completed by July 2018 after which the control system development can begin.

The mathematical modelling is done using forward and inverse kinematics to find the position and orientation of the end effector in relation to the joint angles of the motors. Differential kinematics is then used to determine the velocity and angular velocity of the end effector in relation to the angular velocity of the joints. A dynamic model of the leg is to be produced to relate the end effector forces to the joint torques. This produces a model of the leg's parameters with respect to a local leg coordinate system which can then be used to achieve a desired motion of the hexapod.

Once the control algorithm of a single leg is completed and tested, the modelling of the robot as a whole can begin. This includes the body kinematics which translates the mathematical model in the local leg coordinate system to a model in a global hexapod coordinate system. This way each leg is a complete, modular unit and once several legs are added together they form a full robotic system.

The finished leg unit with control algorithms tested is planned to be complete by the end of 2018. This will allow for sufficient time in 2019 to aggregate the legs' control into a complete control algorithm for the hexapod.

The control algorithm will be programmed using C++. This is done so that it is capable of running on an ARM STM micro-controller. Distributed intelligence will be explored. The aim is to have a separate micro-controller for each leg to process the individual leg kinematics and a master micro-controller processing the hexapod's overall motion and handling remote control with the host computer and the processing of IMU data and communicating with each leg to achieve a desired robot trajectory.

An IMU will be explored and a controller will be developed to allow for attitude control of the hexapod. Implementing an attitude controller will allow the hexapod to maintain a specific attitude irrespective of the terrain over which it walks.

Testing of each control algorithm will be done to ensure that they are functioning as expected. This

will be done by giving the robot a desired trajectory and measuring the accuracy and precision of the movement. The algorithm will be adjusted until the hexapod is able to walk with a smooth leg motion while keeping its body stabilised.

1.5 Anticipated Outputs

The outputs of this project include the production of this dissertation which details the research done and the findings of the project.

A control algorithm is to be developed and software designed around this to allow the hexapod to walk smoothly with a fixed body attitude while being remotely controlled. This software will be implemented into the hexapod system and testing will be conducted to verify that the control algorithms functions effectively.

A GUI is to be developed to run on a host computer which will communicate with the hexapod. This will be designed to allow for remote control of the hexapod walking and show feedback to the user about the hexapod's current state.

The physical hexapod is to be improved and the hexapod robot is the final output of this project.

1.6 Potential Impacts

There is no direct socio-economic impact in the research of a hexapod robot, however the possible applications of the robot include search and rescue. This robot could be used to save lives in difficult situations and aid search and rescue workers, such as mountain rescuers, in doing their job. It allows for rescuers to distance themselves from the dangers associated with search and rescue by sending in the hexapod to assess the surroundings and locate survivors instead of rescuers going in themselves. This lowers the risk to human life and well being.

The research into the mathematical modelling of the hexapod's legs could also be used in prosthetics. The hexapod legs are made up from joints and links following an anthropomorphic manipulator. This is the same configuration as the leg of a human and as such the mathematics is transferable.

2 Background Research

In the current world wheeled robots have dominated locomotion due to their simplicity of design and ease of control. This allows for the manufacture of cheap, efficient and fast vehicles which are used daily. In order for a wheeled vehicle to function correctly it must be used on a relatively smooth surface. Cars for instance, are driven on a paved surface and can only overcome small obstacles, such as speed bumps and pot holes. If the terrain over which the vehicle must traverse is uneven, railed or tracked vehicles can be used. Tracked vehicles, such as tanks, have an increased mobility over complex terrain when compared to wheeled vehicles, however they are still not able to overcome large obstacles. Railed vehicles, such as trains, must have a permanent rail laid down over the entirety of its path. This limits its movement to that of the path of the rail only. Even specially designed all-terrain vehicles can only overcome small obstacles at the cost of a high energy consumption [9]. It is said that over 50% of the Earth's ground surface is inaccessible to wheeled vehicles and for this reason legged or walking robots are a focus of research [10].

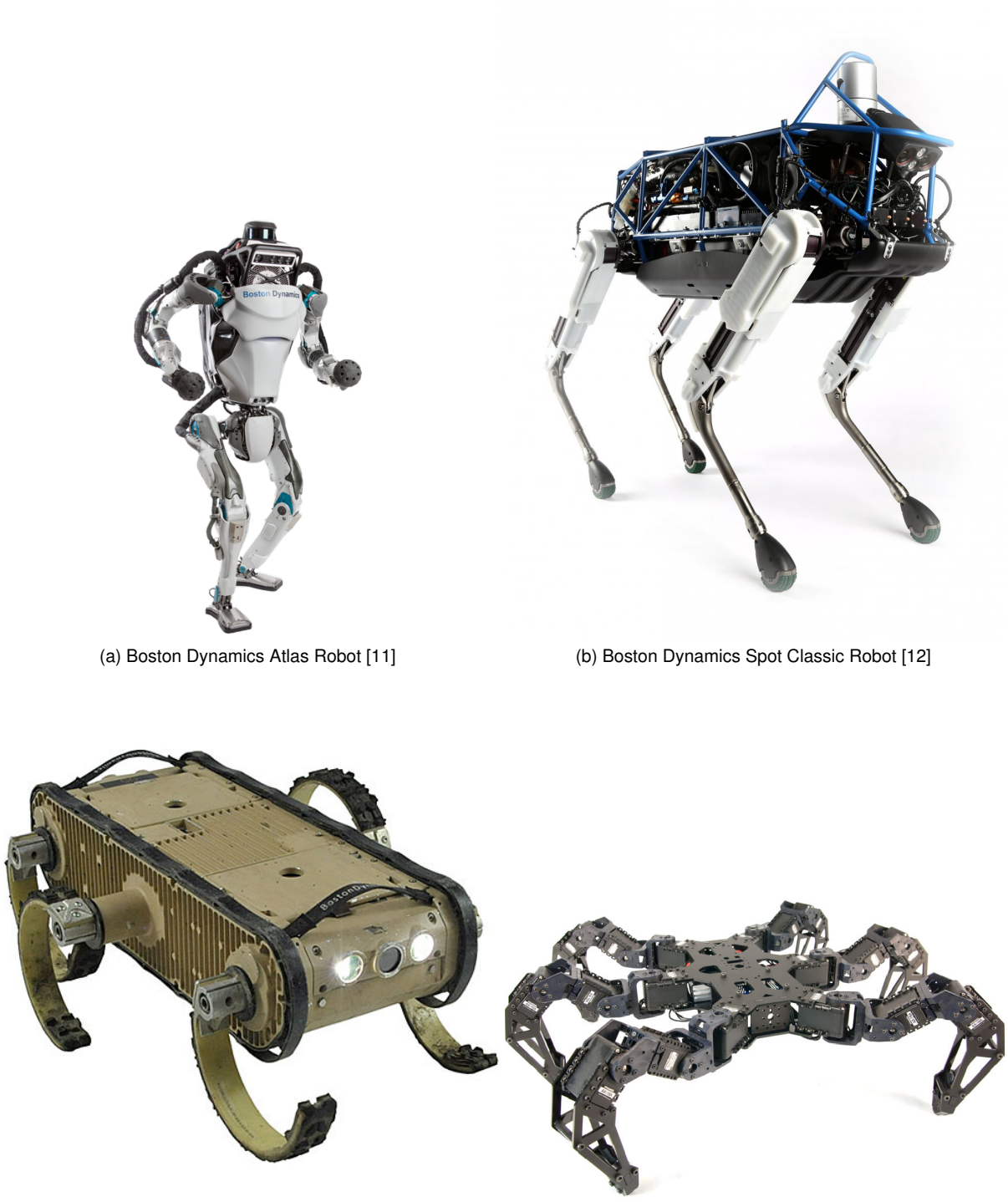
2.1 Walking Robots

Autonomous robots are split into two categories, namely mobile robots and manipulation robots [9]. The focus of this project is on legged mobile robots however the principles and mathematical formulae can be applied to either type of robotic system.

Legged mobile robots are able to access places a wheeled or tracked robot cannot. They have an high degree of mobility due to the flexible nature of their motion. Generally, their legs are made up of a number of links attached to one another with joints where each joint is rotated by an electric motor. By manipulating the motors one can control the position and orientation of the leg and the end effector[5] as well as the velocity, acceleration and end effector forces. This precise motion control allows for the positioning of legs through a sequence of points in such a way to allow a robot walk over rough or uneven terrain. It can lift its legs over obstacles and even use its legs to interact with objects.

[5]The object at the end of the last link of a serial chain (i.e. a foot or gripper)

Some examples of various different types of walking robots can be seen in figure 2.1 below.



(a) Boston Dynamics Atlas Robot [11]

(b) Boston Dynamics Spot Classic Robot [12]

(c) Boston Dynamics RHex Robot [13]

(d) Hexapod PhantomX AX Robot [14]

Figure 2.1: Different Walking Robots

Walking robots have more advantages than simply being able to move over complex terrain. If a (non-biped) legged robot is designed with more than four legs it is capable of losing the functionality of one or more of its legs while retaining the ability to walk.

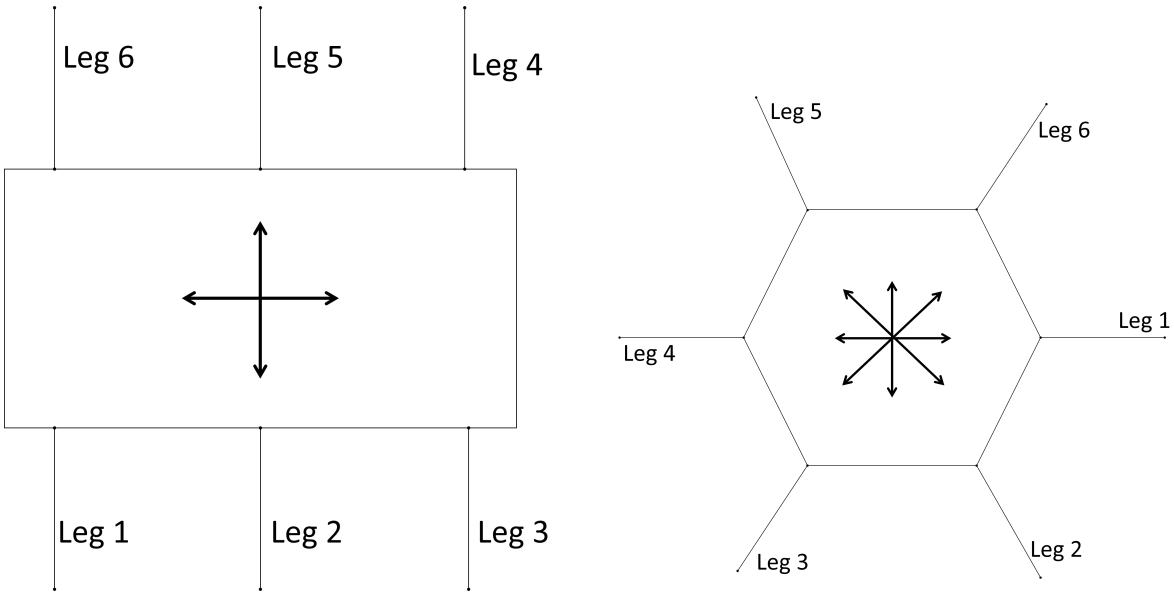
Although walking robots have many advantages over wheeled and tracked robots, they have their disadvantages too. One notable disadvantage is their low speed when compared to wheeled robots due to the complex motion required to propel the robot in a particular direction. Due to the complex motion required to make a legged robot walk, the control algorithm is inherently complicated. This leads to longer development times and more computationally intensive control software [10].

The mechanical design of walking robots is often complicated. Multiple links must be joined to one another through actuators. These links are often made up of complex shapes and curves. Each degree of freedom in a leg of a walking robot requires its own actuator to move it. This means that a legged robot needs far more actuators than a wheeled robot would need which leads to a higher power consumption and higher cost than wheeled robots. This creates an additional problem of developing a high current, light and mobile power supply which can supply power to the robot while it walks [10].

2.2 Hexapod Robots

Hexapod robots are mobile legged robots that walk on six legs. Since non-bipedal a robot can be statically balanced on only three legs and walk with only four legs the hexapod robot has a high degree of flexibility in terms of the motion it can achieve. If any of the six legs fail, the hexapod can continue to operate by adjusting its walking motion to account for the missing leg. Furthermore, not all six legs are needed for stability, this means that while walking legs can be used for other purposes such as interacting with objects [15].

Hexapod robots generally come in one of two shapes; rectangular or hexagonal. In the rectangular shape there is a set of three legs on either side of the long edge of the rectangle. The hexagonal shape hexapod robot has a hexagonal or circular body with the legs distributed around this body evenly. A comparison between the two shapes can be seen in figure 2.2 below.



(a) Rectangular Hexapod Configuration

(b) Hexagonal Hexapod Configuration

Figure 2.2: Hexapod Leg and Body Configurations

It is apparent from the figure above that each design requires different control algorithms to operate and has varying advantages and disadvantages. For example, the rectangular hexapod performs better than the hexagonal hexapod while walking forward and backward, however the hexagonal hexapod is better at turning on the spot and has uniform performance in all directions [16].

The gait of a robot, or animal for that matter, is defined as the cyclic motion pattern that produces locomotion through contact with the ground. The gait can vary in different ways, such as the timing between legs, known as the duty factor, the order in which the legs come into contact with the ground,

the trajectory each leg moves through and more [17].

According to a mathematical analysis of the walking gait of both designs the hexagonal hexapod has a better turning ability, higher stability margin and larger stride length capabilities than the rectangular hexapod [16].

2.2.1 Leg Phases

In order to achieve a walking motion a hexapod robot must lift each foot off the ground, move it through the air to a new position, place it back on the ground at that position and move it once more to pull the hexapod in a particular direction.

The motion during the air is called the swing phase and the motion when the foot is touching the ground is called the support phase.

Support Phase

The support phase of a hexapod's walking motion consists of the time at which the foot touches the ground to the point at which it moves off the ground. During this motion the body of the hexapod is pulled by the leg which propels it in a direction. This is the point at which the load of the hexapod's body and cargo is applied to the leg's links and motors.

Swing Phase

The swing phase of a hexapod's walking motion consists of the time at which the foot lifts off the ground to the time at which it touches back down in a new location. This motion must be carefully designed so that the leg or foot does not come into contact with any other parts of the hexapod, or anything in the outside world, while moving. It is ideally a single smooth motion and can most easily be described by either a portion of a sine wave or a polynomial.

Combined Swing and Support Phase Motion

By combining the above two phases of motion the full motion of the leg is achieved. Figure 2.3 below shows a leg moving through the swing and support phases during horizontal hexapod motion in the x direction.

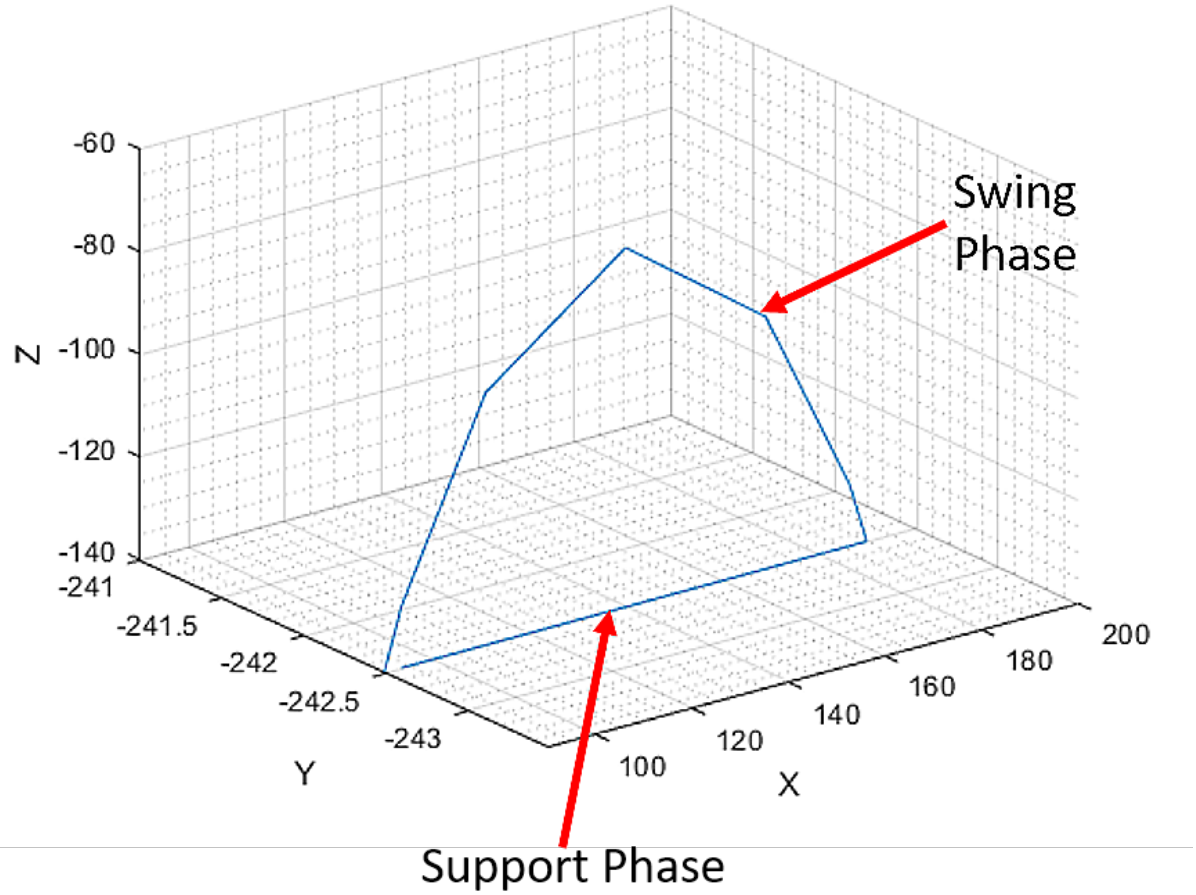


Figure 2.3: Combined Foot Trajectory

It can be seen above that, in this particular path, the stride length is 100mm in the x direction and the step height is 50mm in the z direction. Note that the z axis is shown from -60mm to -140mm because the zero point is the body of the hexapod and down is negative. As such to reach the ground if the hexapod is standing at a height of 140mm above the ground, the leg must move to a z position of -140mm. The coordinates in the plot above are relative to the center of the hexapod's body.

2.2.2 Walking Gaits

The walking motion that a hexapod uses is called a gait. It defines the order in which it moves its legs and greatly affects both the style and efficiency of walking. The gaits investigated here include the free gait, the tripod gait, the wave gait and the quadruped gait. A specialised gait to be used when gripping an object is also investigated.

One cycle of the gait includes both the swing and the support phase and starts and ends with the leg at the same position. The duty factor of the leg is defined as the fraction of the gait cycle during which the leg is in the support phase [18].

Free Gait

The free gait is characterized as having no periodic or cyclical pattern. It moves each leg independently and for this reason it is terrain adaptive. Each leg's foot hold and swing motion is planned using a flexible algorithm that takes the ground features and the robots position and orientation into account [19]. It is a more efficient walking gait than others over complex terrain however it requires more advanced control and sensors to input data about the robots surroundings. Using knowledge about its environment the robot is able to determine acceptable positions to place its feet relative to the ground features and by doing this for all legs walking is achieved.

Tripod Gait

The tripod gait is characterized as having three legs on the ground in the support phase while the other three legs are in the air in the swing phase. In each gait cycle the body of the hexapod moves two strides. The duty factor of this gait is $\frac{1}{2}$.

This is the fastest hexapod gait of those discussed and as demonstrated in [20] the free gait algorithm on flat ground spontaneously produces a tripod gait.

The foot fall pattern of the tripod gait can be seen in table 2.1 below.

Table 2.1: Tripod Gait Foot Fall Pattern

Leg 1	■		■		■	
Leg 2		■		■		■
Leg 3	■		■		■	
Leg 4		■		■		■
Leg 5	■		■		■	
Leg 6		■		■		■

In the foot fall table above the dark cells represent a leg in the swing phase while a light cell represents a leg in the support phase.

Wave Gait

The wave gait is characterized as having five legs in the support phase while one leg is in the swing phase. The duty factor of this gait is $\frac{5}{6}$. Having five legs in contact with the ground at a time makes this gait the most stable, however it is the slowest of the gaits discussed [21]. The foot fall pattern of the wave gait can be seen in table 2.2 below.

Table 2.2: Wave Gait Foot Fall Pattern

Leg 1	■					
Leg 2		■				
Leg 3			■			
Leg 4				■		
Leg 5					■	
Leg 6						■

In the foot fall table above the dark cells represent a leg in the swing phase while a light cell represents a leg in the support phase.

Hexapod Quadruped Gait

The quadruped gait is characterized as having four legs in the support phase while two legs are in the swing phase. This gait is a combination of speed and stability as four legs are on the ground at all times. The duty factor of this gait is $\frac{2}{3}$ [19]. The foot fall pattern of the quadruped gait can be seen in table 2.3 below.

Table 2.3: Hexapod Quadruped Gait Foot Fall Pattern

Leg 1	■			■		
Leg 2		■			■	
Leg 3			■			■
Leg 4	■			■		
Leg 5		■			■	
Leg 6			■			■

In the foot fall table above the dark cells represent a leg in the swing phase while a light cell represents a leg in the support phase.

Gripping Gait

This gripping gait is used when two of the legs are re-purposed as grippers. This turns the hexapod into a quadruped and thus the gait must be adjusted to that of a quadruped [5]. The quadruped gait used for hexapods is the crawl gait. This is characterized as having three legs in the support phase while one leg is in the swing phase. The foot fall pattern of the gripping gait can be seen in table 2.4 below.

Table 2.4: Gripping Gait Foot Fall Pattern

Leg 1						
Leg 2						
Leg 3						
Leg 4						
Leg 5						
Leg 6						

In the foot fall table above the dark cells represent a leg in the swing phase while a light cell represents a leg in the support phase and the red cells represents legs being used as grippers.

Comparison of Gaits

A comparison of the gaits discussed above can be seen in table 2.5 below. Each parameter is given a rating from $1-5$ so for example a 0 for speed means that it is slow, a 5 for stability means that it is very stable and a 0 for complexity means that it is simple to implement. Reasons for each value given can be found in Appendix A.

Table 2.5: Hexapod Gait Comparison

Gait	Duty Factor	Speed	Stability	Preferred Surface Type	Interact with Objects	Control Complexity
Free	N/A	5	5	Any	No	5
Tripod	$\frac{1}{2}$	4	2	Smooth	No	0
Wave	$\frac{5}{6}$	0	4	Smooth	No	1
Quadruped	$\frac{2}{3}$	2	3	Smooth	No	2
Gripping	$\frac{3}{4}$	1	1	Smooth	Yes	3

2.2.3 Stability Margin

One of the main goals in having a walking robot navigate complex terrain is for the robot to do so quickly while remaining stable. It is advantageous if the base of the robot remains as horizontal as possible in order to allow the carrying of a sensor payload (such as cameras) and/or a cargo load. This can only be done to the point at which the robot remains stable, after which stability will remain the priority so that the robot does not fall over.

The stability margin of the robot is defined as the shortest distance from the center of mass of the robot to the lines joining its feet in the support phase. The smallest of these distances is the stability margin which must remain above a certain value, as defined by the user of the hexapod [22]. The stability triangle can be seen in the image below where the stability margin is the shortest of green lines (in this case they are all the same length as the center of mass of the hexapod is in the center of its body).

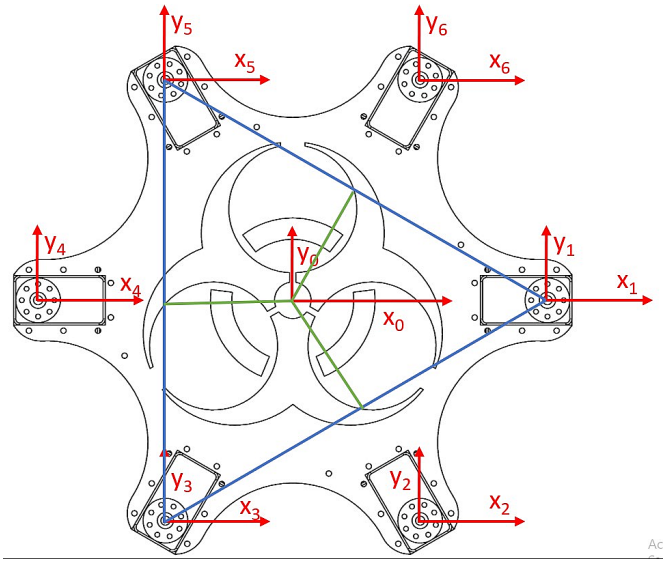


Figure 2.4: Stability Triangle

The configuration of the robot above assumes that legs 1, 3 and 5 are supporting the robot while the other 3 legs are off the ground. The blue lines form the stability triangle while the green lines denote the 3 stability margins.

In the robot configuration above there are only 3 feet on the ground therefore the stability polygon is a triangle, however in other cases any number of feet could be touching the ground therefore increasing the number of sides of the polygon [22] and the number of stability margin values to compare to find the lowest one.

2.3 Biomimicry

Reductive biomimicry is the process of studying and learning from nature and applying its principles to design and engineering [23]. This can lead to new and exciting products and advances in technology. The interest in biomimicry here is to do with the configuration of legs and walking motion of insects. Of particular interest is the common robotic manipulator configuration known as the anthropomorphic manipulator. This manipulator configuration is based on the human leg. It can be easily made up of three revolute joints as is the case of the three dynamixel servo motors in each leg of the hexapod.

2.4 Distributed Intelligence

Distributed intelligence is the process where multiple devices work together as one to overcome obstacles and solve problems. If a robotic system requires multiple complex tasks performed at the same time it can be beneficial to have various different modules accomplishing each task [24]. This decreases the computational requirements for each module by splitting the load between multiple, separate modules.

In the case of the hexapod, each leg must calculate its own inverse and differential kinematics for every point it wishes to move to. This is a computationally intensive calculation and if one module has to calculate the kinematics for all six legs there could be a lag time while moving between points. Although this lag time may not be significant enough to warrant six different control modules (one for each leg) at the moment, as the functionality of the hexapod increases and more sensors are added this lag time may increase further and become an issue.

The use of a different ‘brain‘ for each leg allows for separation of the legs into modules, it does however introduce a problem with syncing the timing of all of the legs. The Dynamixel motors already fitted on the hexapod use a function called sync write [25]. This function allows for broadcasting a serial command to all the motors at once which allows them all to be controlled, in parallel, at precisely the same time.

Separating the legs into different modules means that each module must communicate with one central controller before communicating with the motors. This adds another level of communication that is required for the hexapod to operate and could therefore introduce a lag time that may be greater than when there is not distributed intelligence.

3 2018 Hexapod Structure (MK1)

The version of the hexapod as it was at the start of 2018, referred to as MK1, was designed as a symmetrical, circular robot by students at the University of Cape Town's Robotics and Agents Research Lab [4], [5], [6]. It is shown in figure 3.1 below.

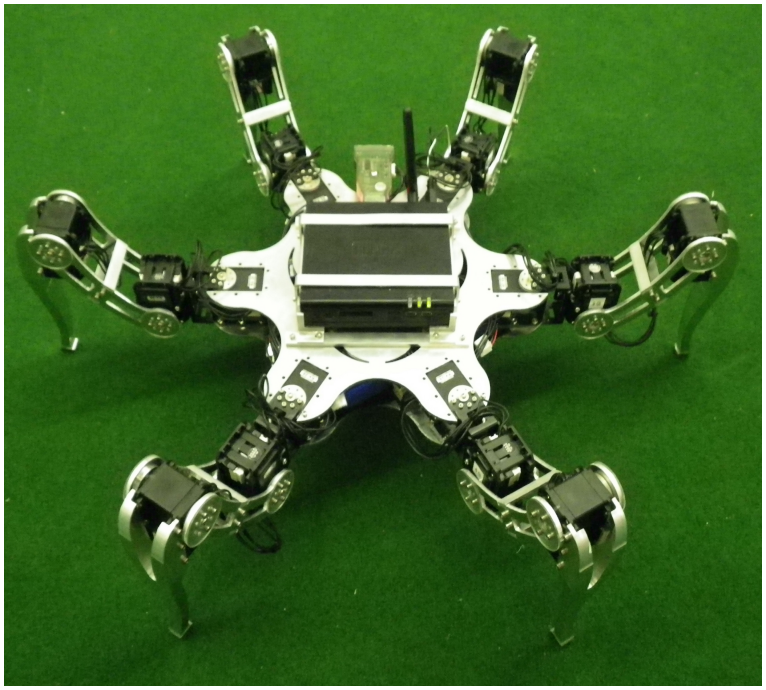
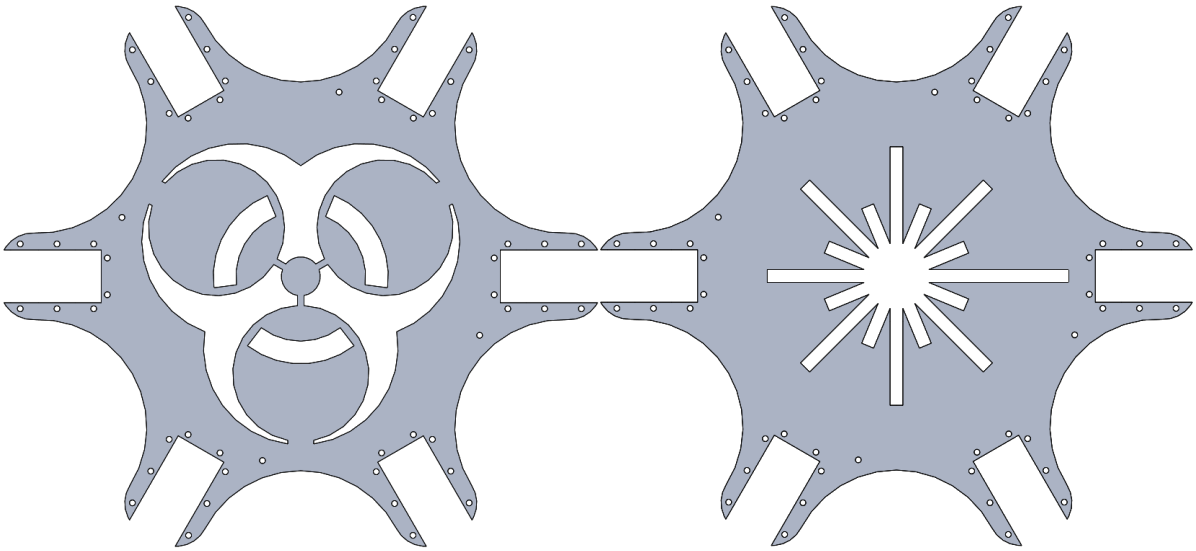


Figure 3.1: Current UCT Hexapod Robot

3.1 Body

The body is made from two 3mm thick aluminium plates between which the six horizontal hip motors are mounted. The top view of the top and bottom plates of the body can be seen below.



(a) Top Plate (b) Bottom Plate

Figure 3.2: Hexapod Body Plates Top View

3.2 Legs

Each leg is made up of three dynamixel motors connected to one another through a serial chain of $3mm$ thick aluminium links. The final, third leg link is a solid aluminium profile. A 3D printed foot is attached to the end of the third link. This was done so that a force sensitive resistor (FSR) could be mounted into the foot [4]. By measuring the voltage across the FSR in each foot the controller is able to determine whether or not the foot was touching a surface. The leg structure can be seen in figure 3.3 below.

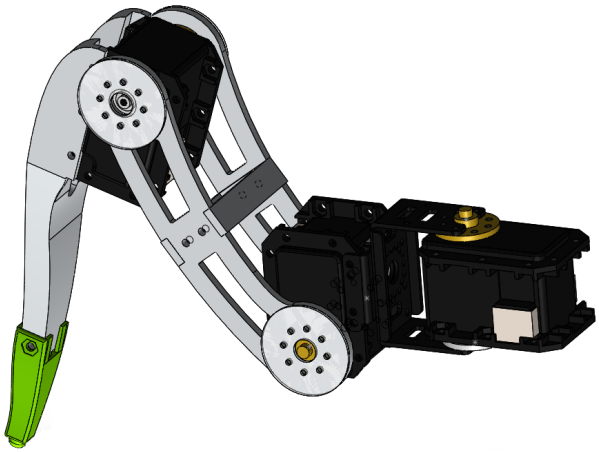


Figure 3.3: Hexapod Leg

3.3 Motors

The motors which are responsible for actuating each joint of the hexapod are Dynamixel RX-28 servo motors. They have built-in controllers and are controlled using serial communication. Each motor has a unique ID and is connected to the next motor in series, forming a daisy chain system. When the controller sends out a command all motors receive that command but only the motor that the command is meant for will act on it.

Sending commands to each motor allows the setting of motor parameters, such as maximum torque and temperature limits, and the reading of parameters, such as current load and motor voltage.

To operate the motors the controller sends a goal position and speed to which the motors respond by moving to said position. A movement command (or in fact any command) can be sent to all motors at the same time using a function called sync-write [25]. This allows for control of all 18 motors at precisely the same time which is crucial for a smooth walking motion.

A full list of the control table containing all the registers of the motor can be found in Appendix B.

3.3.1 Serial Communication

Serial communication uses two pins to communicate. A receive pin (RX) and a transmit pin (TX). The serial module can communicate in two different modes, full duplex and half duplex. Full duplex is a communication mode whereby the controller can send and receive data at the same time while half duplex allows for transmission in only one direction at any given time. The dynamixel motors communicate using half duplex and as such half duplex mode must be selected when attempting to control the motors.

Communication between the motors and the controller requires that both the motors and the controller are set to communicate at a defined speed, called the baud rate. The baud rate can range anywhere from 9,600 to 1,000,000 bits per second [26].

Each motor has a specified identify number (ID) which is referred to when trying to communicate with a particular motor.

The dynamixel motors receive various commands from the controller in the form of an instruction packet and respond to the controller with a status packet. The instruction packet can contain any one of the following commands [25]:

Value	Name	Function	No. of Parameters
0x01	PING	No execution. It is used when controller is ready to recevie Status Packet	0
0x02	READ DATA	This command reads data from RX-64	2
0x03	WRITE DATA	This command writes data to RX-64	2 or more
0x04	REG WRITE	It is similar to WRTE_DATA, but it remains in the standby state without being executed until the ACTION command arrives.	2 or more
0x05	ACTION	This command initiates motions registered with REG WRITE	0
0x06	RESET	This command restores the state of RX-64 to the factory default setting.	0
0x83	SYNC WRITE	This command is used to control several RX-64s simultaneously at a time.	4 or more

Figure 3.4: Dynamixel Instructions

The data is transmitted to the motors one byte at a time and the status packet is returned to the controller after a delay defined by the value of the 'Return Delay Time' register in the motor. This gives the controller and circuitry time to switch from write mode to read mode in order to receive the status packet.

An example instruction and status packet when the controller attempts to read the current internal temperature of a motor with an ID of 1 can be seen below [25]. Note that this extract is from the dynamixel RX-64 datasheet however the communication is the same for RX-64 motors and RX-28 motors.

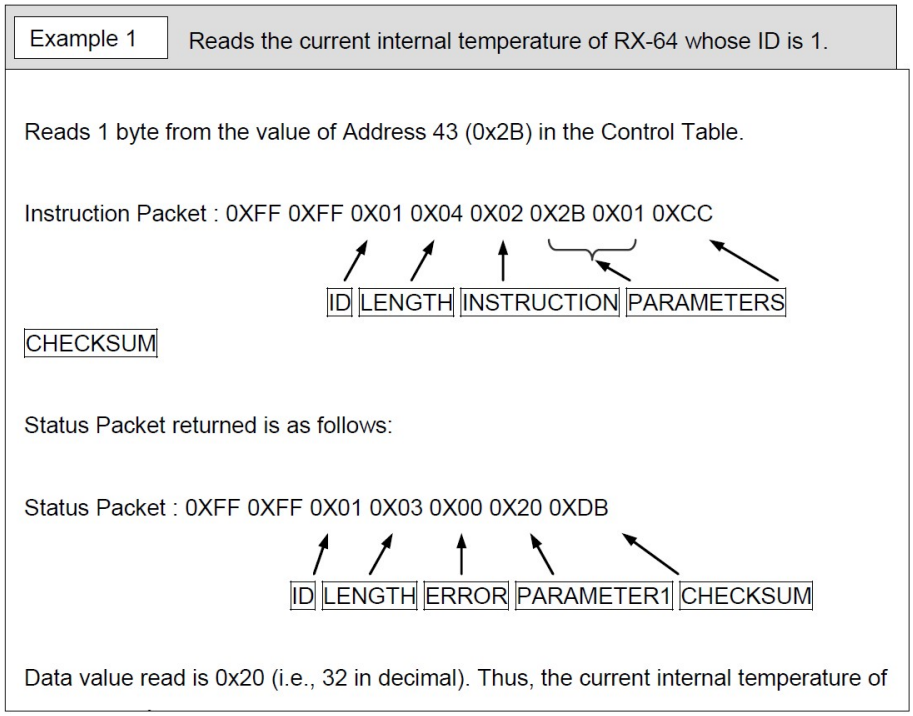


Figure 3.5: Dynamixel Example Communication Packets

3.3.2 Control Loop

The dynamixel motors have a built in PID control loop for position control [27]. The clockwise and counter-clockwise compliance margin and slope of this control loop can be set using registers 26 to 29 of the dynamixel control table. This is done to adjust the pattern of output torque to improve shock absorption and smooth motion. The compliance margin is the position around the goal position where the output torque is zero and the compliance slope is the position around the goal position where the output torque is reduced. The wider the compliance slope the smoother the motion is, however if it is too wide there will be an error between the commanded position and the actual position of the motor [25].

3.4 Controller

The controller is a FitPC-2 fanless computer. It allows for the use of a LabVIEW [28] VI running on Windows 7 to control the hexapod. This was advantageous at the time due to the simplicity of developing control software in LabVIEW.

Figure 3.6: FitPC 2 Hexapod Controller [29]

The FitPC uses a USB2Dynamixel serial to USB converter to connect with the motors. This devices converts the USB communication protocol into the RS485 protocol required by the motors to operate. It can be seen in figure 3.7 below [25].

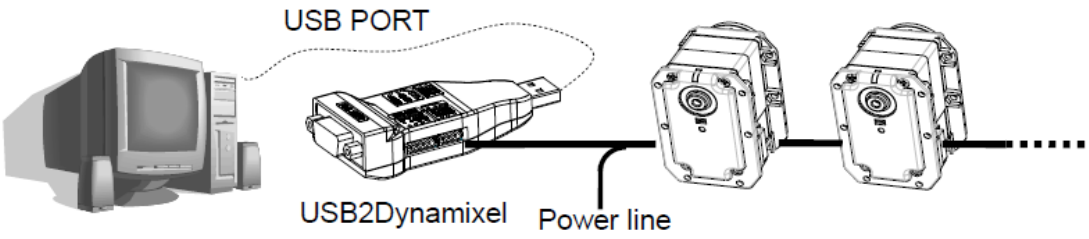


Figure 3.7: USB2Dynamixel Converter

3.5 Software

The LabVIEW program facilitated communication between the FitPC and the motors which allowed for motion. A screenshot of the LabVIEW program can be seen in figure 3.8 below.

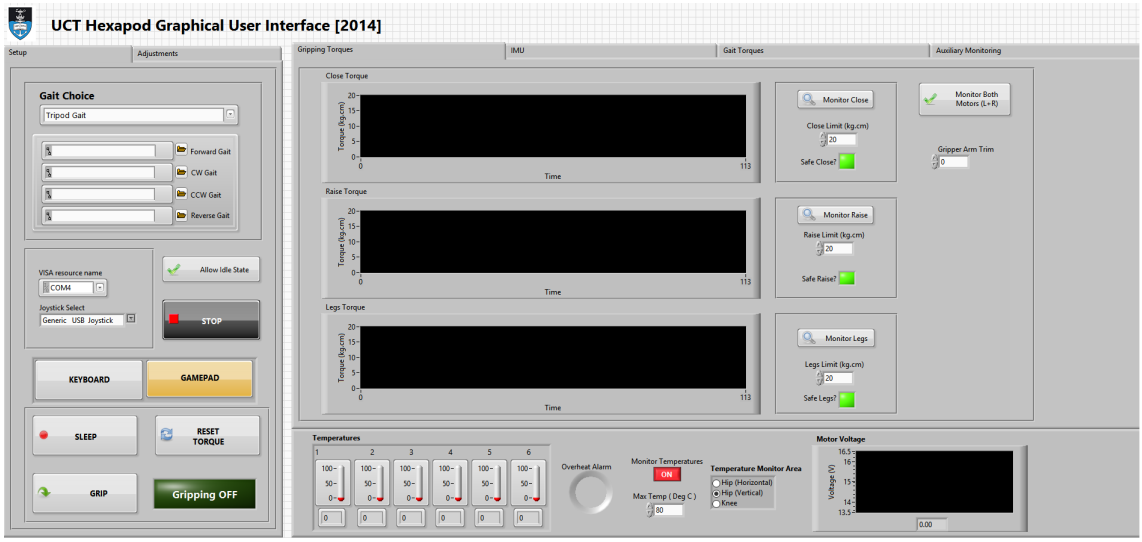


Figure 3.8: LabVIEW Control VI

This above LabVIEW [28] VI reads in a file of motor positions and writes them out to the serial port of the computer in sequence, achieving a walking motion. It allows for the selecting of multiple different gaits and has the ability to measure and record the torques and temperatures of all motors.

3.6 Overall Functionality and Capabilities

The MK1 hexapod was capable of walking with various different gaits being read in from a file. By manipulating the data in the file the body height and step height of the hexapod could be adjusted. The hexapod could walk forward, backwards, left and right and could turn on the spot.

It had a gripping gait in which it would re-purpose its two front legs to become grippers and adjust the other four legs to walk only using those four. The gripping legs could clutch objects and move it up and down and towards itself or further away from itself.

IMU data could be monitored as well as torque, voltage and temperature information of each motor.

It could be controlled through the FitPC using either a keyboard or a joystick.

3.7 Observed Problems

During operation of the 2018 hexapod multiple problems were discovered. These problems were detailed so that they can be a focus of this project.

1. The hexapod was controlled by a computer running Windows 7. This computer took a long time to boot up and often failed to do so correctly. If power to the hexapod was momentarily lost during operation the computer would have to reboot which would again take up to 6 minutes.
2. The computer did not have a built in fan which made it light, however the LabVIEW VI controlling the hexapod is computationally intensive and the computer would, on occasion, overheat and shutdown.
3. The LabVIEW VI is slow and often lost communication with the motors which would cause it to crash.
4. If constant reading of torques and temperatures (or any other motor data) into the LabVIEW VI is switched on, the walking smoothness suffers due to delays in communication with the motors.
5. The computer cannot connect to peripheral GPIO devices such as external temperature sensors and LED lighting without additional circuitry.
6. The vertical hip motors often hit their torque limits and would overload, causing them to shutdown and the hexapod's body to fall to the ground.
7. The 3D printed feet broke as the 3D printed layers sheared apart while walking.
8. The wires would get caught in the links as the legs moved and would often get severed and short circuit.
9. The top and bottom plates had shapes cut out of them which were for appearance and not functional purposes. This meant that mounting new objects to the body of the hexapod is difficult as there are limited places to drill mounting holes.

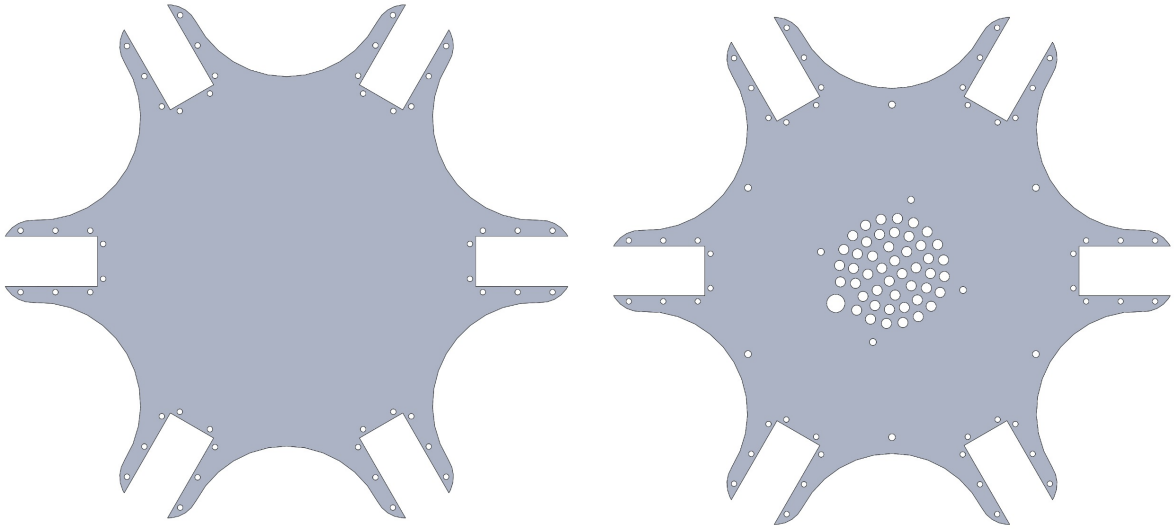
4 Mechanical Design

The goal of this project was to use the current RARL hexapod and develop it further. As such no major changes were made to the mechanical structure of it, however after observing the 2018 hexapod in operation various mechanical inadequacies were noted. Alterations to the hexapod's components were made to address these problems.

4.1 Body

The top and bottom plates of the hexapod had patterns cut into them which served no purpose other than aesthetics. When trying to mount new components to the hexapod these cut outs prevented adding mounting holes for these components.

The top and bottom body plates were remade without patterns or holes in them (aside from those required for mounting components and air flow). The two plates were laser cut from 3mm aluminium. Top view drawings of the plates can be seen in the figure below.



(a) Top (b) Bottom

Figure 4.1: Hexapod Body Plates

4.2 Motors

The vertical hip motors, which held the majority of the weight of the hexapod and had the most torque applied to them out of all other motors, were inadequate and would often overload under the weight of the hexapod. The Dynamixel RX-28 motors can produce a maximum torque of 37.7 kgf.cm while the Dynamixel RX-64 motors can produce a maximum torque of 77.2kgf.cm. This is over double the RX-28. These RX-64 motors were readily available in RARL at UCT as they had already been sourced for this hexapod upgrade. For this reason the six vertical hip motors were replaced with RX-64 motors to prevent overloading under the weight of the hexapod.

The RX-64 motors are slightly wider than the RX-28 motors and as such a larger connector joining the two sides of the leg's link was fitted and perspex spacers were laser cut and added to ensure a proper fit of the motors. The model of the leg with the RX-64 motor mounted can be seen in figure 4.2 below.

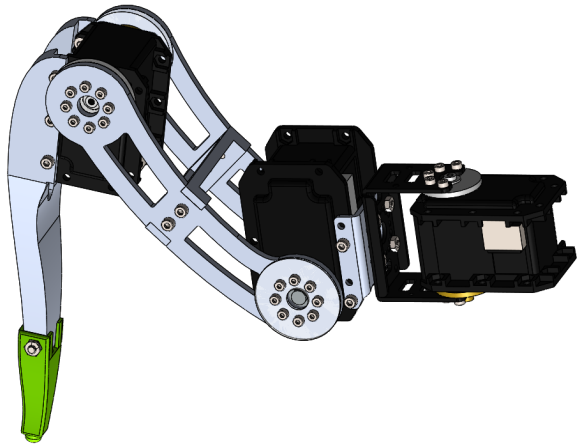


Figure 4.2: Hexapod Leg with RX-64 Motor

4.3 Feet

There is no need for FSR's in the feet of the hexapod to determine whether or not each foot is touching the ground. This is because the torque on each motor can be read by the controller. This allows for foot force determination using the dynamic model of the leg.

Using the motor torques and the dynamic model of the leg is more accurate than using a force sensor at the tip of each foot as it provides a three-dimensional force measurement. It also removes the need for GPIO pins being used and long wires to the end of each foot, which have the possibility of being caught up in the legs as was seen with wiring in the previous design.

This lead to a slight redesign of the foot of each leg. The internal cavity in the foot for the FSR was removed which aided in strengthening the foot. When 3D printing the modified feet, they were printed such that the layers are perpendicular to the foot shear force direction. This also helped in preventing the feet from shearing between layers during operation.

4.4 Leg Calibration

It was noticed that after replacing the motors of the legs that each leg was not lined up with one another. If all motors on all legs were set to the zero position not all legs were in precisely the same position. This was due to slight misalignment when attaching the motor shafts to the leg links.

In order to mechanically align the legs such that they are all in sync, a large perspex protractor was designed. It was laser cut from 5mm perspex and mounted to each leg one at a time as shown in the figure below.

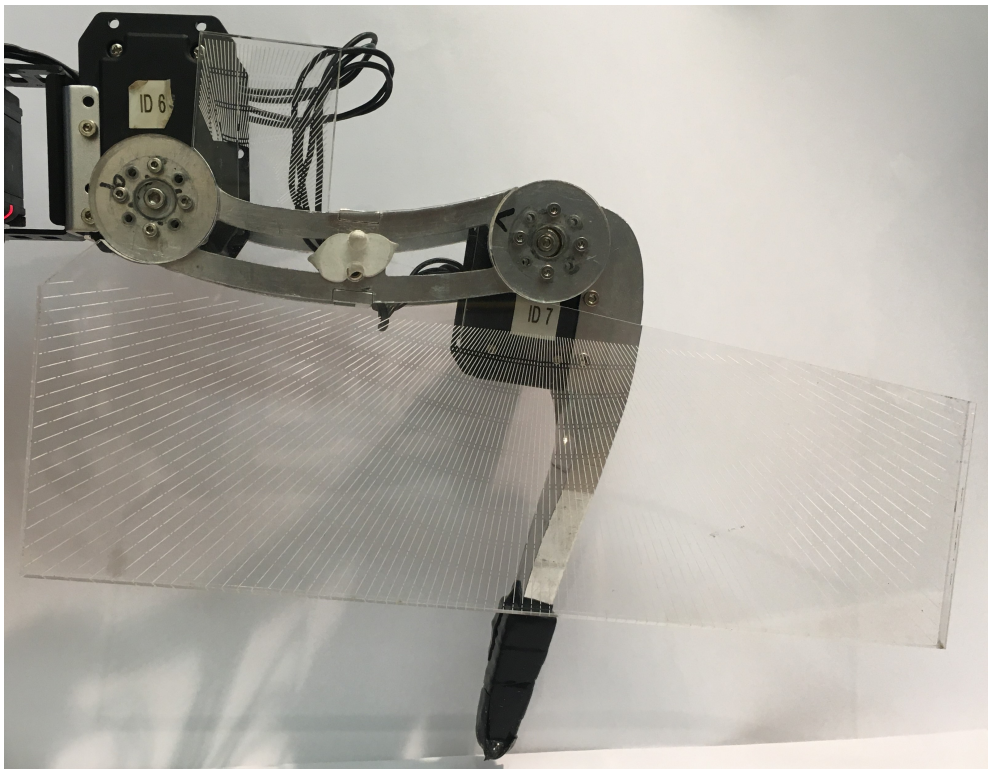


Figure 4.3: Hexapod Leg with Calibration Protractor Mounted

It allowed each link of the leg to be positioned onto the motor shaft accurately. The motors shafts are splined and as such they cannot be mounted to the shafts at any rotation increment. They can only be mounted in increments of the shaft spines, $\pm 18°$ on the RX-28 motors and $\pm 12°$ on the RX-64 motors. The above protractor ensured that each leg was mounted onto the same spline on each motor, ensuring that the zero position for each leg is the same.

4.5 Discussion of Mechanical Design

The replacing of the RX-28 motors with the RX-64 motors allowed for more torque capabilities in the vertical hip joint of the hexapod. Through use of the hexapod it could be seen that this torque increase was sufficient to allow the hexapod to walk correctly under its own weight. The feet of the hexapod would often hook on the surface of the ground and this would cause the torque on the motors to increase, however the RX-64 motors were capable of continuing to function during this time and the motors did not overload.

The 3D printed feet were attached to the hexapod and proved to be stronger than the previous 'MK1' hexapod's feet. None of the new feet of the hexapod failed mechanically by shearing between layer during normal operation of the hexapod over the duration of testing in this project.

The legs were successfully aligned with one another and when all legs were set to the same position, zero radians on all motors, they were all positioned in the same configuration.

It was noticed that when the hexapod was standing still with its weight distributed over all of its legs, some legs were supporting more of the hexapod's weight than others. This lead to the conclusion that even after the mechanical alignment of the legs, they were still not perfectly aligned. This lead to the feet slipping while walking as they were not making contact with the ground properly. This final misalignment needed to be fixed.

This was remedied by adding soft silicon padding to the bottom of each foot which aided in adding grip between the feet and the ground as well as allowing for a more even weight distribution between the feet. When the hexapod stands each of the pads on the feet compresses slightly so that every foot is making contact with the ground. This in conjunction with a software offset to each motor (discussed in the programming section of this report) aided in reducing the slipping of the hexapod's feet.

5 Leg Kinematics

In order to control the motion of a legged robot it is important to understand the relationship between the position and orientation of the leg of the robot and the joint angles which achieve this position. To do this, solutions to the forward kinematics, which translates the joint angles of the leg to the cartesian position of the foot tip, and the inverse kinematics, which translates the cartesian position of the foot tip to the joint angles of the leg, are required [30].

One can find the forward and inverse kinematics of each of the legs of a robot separately and combine them using body kinematics to produce a full model of the robot or one can find the forward and inverse kinematics of the robot as a whole, called the parallel kinematics of the robot [22].

It is advantageous, in this case, to consider each leg separately as it allows for the independent use of each leg and distributed intelligence. It also allows each leg to be thought of as its own module and the mathematics developed can then be applied to any future 3-DOF anthropomorphic walking robot by simply developing the body kinematics for the particular leg configuration of that robot.

To control the speed of the foot tip the differential and inverse differential kinematics are required. This translates the rotational speed of each joint, in rad/s, into the linear speed of the foot tip in m/s and vice versa.

To develop these solutions one must first understand rigid body motion which uses matrices to both translate and rotate between coordinate systems.

Note that the solutions developed in this section are defined relative to a coordinate system attached to each leg as seen in figure 5.1 below.

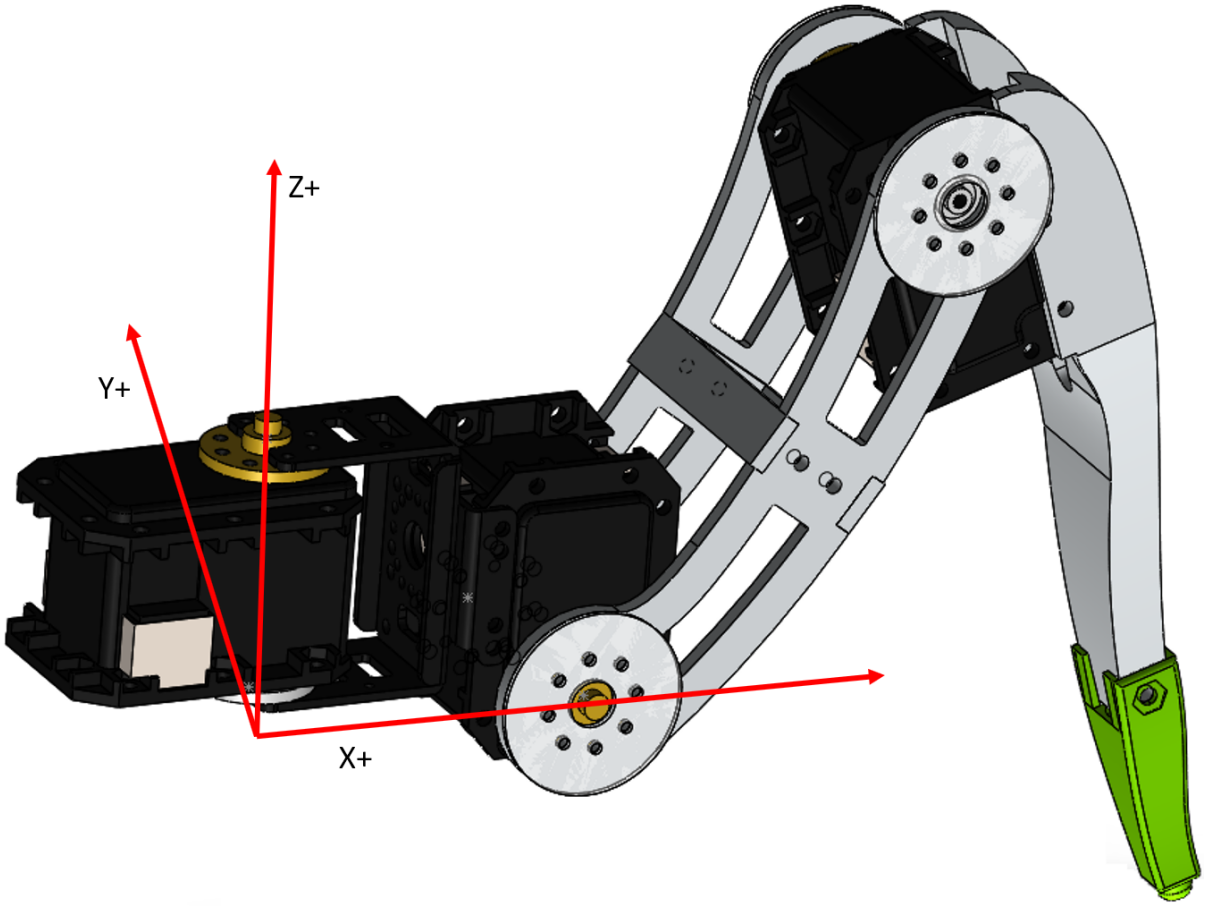


Figure 5.1: Hexapod Robot Leg Coordinate System

The coordinate system is fixed to the shaft of the horizontal hip motor with the positive z-axis concentric to this shaft. The x-axis points in the direction of the next motor in the chain and the y-axis is then assigned accordingly. The origin of the frame is placed at the bottom surface of the horizontal hip motor.

5.1 Rigid-Body Motion

In order for a robotic system to be analysed and mathematically modelled, it must first be defined. In this context, a robot is defined as a series of links joined to one another through rotational or prismatic joints. It is important to understand the relationship between each link. In a serial, open chain robotic system, each link is connected to the previous link and the last link acts as an end-effector such as a gripper or foot. This allows for motion of one link relative to another, producing a desired end-effector motion. It is said to be open chain because the final link does not interact with the robot itself, but with external objects such as the ground [31]. Each link can be placed at a position in space and rotated to a particular orientation. With an appropriate number of links this allows for complete three dimensional motion.

Rigid-body motion is defined as the process of moving and/or rotating an object while preserving the distance between points on the object. The motion must be physically possible and as such, reflections are not included as a reflection would yield a new object [32].

A rotation about a point is represented by an $N \times N$ matrix who's determinate is equal to ± 1 and the following equation holds: $A^T A = I$ for matrix A. Any real matrix which satisfies both of the above conditions is called a special orthogonal matrix, or $SO(N)$ [32]. In the case of a counterclockwise rotation around a fixed axis by an angle θ in three dimensional space the following matrices are produced.

$$\overline{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \end{bmatrix} \tag{5.1}$$

$$\overline{R}_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{5.2}$$

$$\overline{R}_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.3}$$

In order to produce a three dimensional rotation of a rigid body in the direction of each axis through one transformation, the above matrices are multiplied together in the order of the desired rotation order. A commonly used order of rotation in aerospace and engineering is XYZ and is known as Tait–Bryan Angles [33].

The matrix R_{XYZ} can be seen below.

$$\begin{aligned}\overline{R}_{xyz} &= \overline{R}_x(\phi)\overline{R}_y(\theta)\overline{R}_z(\psi) \\ &= \begin{bmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ cos(\psi)sin(\theta)sin(\phi) - cos(\phi)sin(\psi) & cos(\phi)cos(\psi) + sin(\theta)sin(\phi)sin(\psi) & cos(\theta)sin(\phi) \\ cos(\phi)cos(\psi)sin(\theta) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - cos(\psi)sin(\theta) & cos(\theta)cos(\phi) \end{bmatrix}\end{aligned} \tag{5.4}$$

The matrix above represents a three dimensional rotation using Euler Angles, ϕ represents a rotation angle about the x-axis, θ represents a rotation angle about the y-axis and ψ represents a rotation angle about the z-axis. These rotations are known as roll, pitch and yaw respectively.

Combining the above three dimensional rotation with a translation in each axis allows us to achieve complete three dimensional rigid-body transformation. It yields a position vector for the objects position and a rotation matrix for the objects orientation as seen in the matrix below.

$$\overline{H} = \begin{bmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) & P_x \\ cos(\psi)sin(\theta)sin(\phi) - cos(\phi)sin(\psi) & cos(\phi)cos(\psi) + sin(\theta)sin(\phi)sin(\psi) & cos(\theta)sin(\phi) & P_y \\ cos(\phi)cos(\psi)sin(\theta) + sin(\phi)sin(\psi) & cos(\phi)sin(\theta)sin(\psi) - cos(\psi)sin(\theta) & cos(\theta)cos(\phi) & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.5}$$

This leads to a rigid body rotation about the x, y and z axis of angles ϕ, θ and ψ respectively as well as a simultaneous translation in the x, y and z axis of distances P_x, P_y and P_z. This matrix is known as a homogeneous transformation matrix [32].

5.2 Exponential Coordinates

In the previous sections the rotation matrices are represent in the Euler Angle notation in which a three dimensional rotation matrix is represented as a product of three rotations. Exponential coordinates are represented by a single rotation axis (unit vector ω) and an angle of rotation (θ). This exponential notation comes from linear differential equations [31] and it is useful in describing screw motions.

5.2.1 The Skew Symmetric Matrix

A matrix S is said to be skew symmetric if $S^T + S = 0$ [34]. This leads to the following skew symmetric matrix S for the general case of a 3×3 matrix A.

$$A = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \tag{5.6}$$

$$S(A) = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \tag{5.7}$$

The skew symmetric matrix possess many useful properties which will further simplify future calculations [34].

1. The operator S is linear, i.e. $S(\alpha a + \beta b) = \alpha S(a) + \beta S(b)$ for any vectors a and b belonging to R^3 and scalars α and β.
2. For any vectors a and p belonging to R^3, $S(a)p = a \times p$ where $a \times p$ denotes the vector cross product.
3. If $R \in SO(3)$ and a, b are vectors in R^3 it can be shown that $R(a \times b) = Ra \times Rb$. This equation is not true in general unless R is orthogonal. It says that if we first rotate the vectors a and b using the rotation transformation R and then form the cross product of the rotated vectors Ra and Rb, the result is the same as that obtained by first forming the cross product $a \times b$ and then rotating to obtain $R(a \times b)$.

4. For $R \in SO(3)$ and $a \in R^3$, $RS(a)R^T = S(Ra)$ This property follows easily from the previous equations as follows. Let $b \in R^3$ be an arbitrary vector. Then $RS(a)R^T b = R(a \times R^T b) = (Ra) \times (RR^T b) = (Ra) \times b = S(Ra)b$ and the result follows. The equation says therefore that the matrix representation of $S(a)$ in a coordinate frame rotated by R is the same as the skew symmetric matrix $S(Ra)$ corresponding to the vector a rotated by R.

5.2.2 Deriving Exponential Coordinates

Starting with the following vector linear differential equation:

$$\dot{x}(t) = Ax(t) \tag{5.8}$$

where $x(t) \in R$, $A \in R^{n \times n}$ is a constant and the initial conditions of $x(0) = x_0$ apply.

The solution to equation 5.8 is as follows.

$$x(t) = e^{At}x_0 \tag{5.9}$$

Substituting equation 5.9 into 5.8 yields the following solution to the differential equation.

$$\dot{x}(t) = Ae^{At}x_0 \tag{5.10}$$

Suppose a vector $\overline{p}$ is rotated by an angle θ around an axis ω at a constant rate of 1 rad/s. The differential equation is then given by the following [35].

$$\dot{p} = \hat{\omega} \times p \tag{5.11}$$

An alternative way of representing the cross-product between two vectors is by using the skew symmetric matrix defined previously. For a vector $\omega \in R^3$, $[\omega]$ is defined as the skew-symmetric matrix of ω [31].

$$[\omega] = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{5.12}$$

The following is known from property 2 about skew symmetric matrices discussed previously.

$$p \times \hat{\omega} = p[\hat{\omega}] \tag{5.13}$$

For the three dimensional vector $\omega \in R^3$ and the rotation matrix $R \in SO(3)$ the following holds:

$$R[\omega]R^T = [R\omega] \tag{5.14}$$

Continuing from equation 5.9, substituting in θ for t and using equation 5.11 the following equation is developed.

$$p(\theta) = e^{[\hat{\omega}]\theta} p(0) \tag{5.15}$$

A closed-form solution for $e^{[\hat{\omega}]\theta}$ can now be found. It can be shown that $[\hat{\omega}]^3 = -[\hat{\omega}]$ for the unit vector $\hat{\omega}$ [31].

Expanding the matrix exponential $e^{[\hat{\omega}]\theta}$ using the Taylor Series expansion and replacing $[\hat{\omega}]^3$ with $-[\hat{\omega}]$, $[\hat{\omega}]^4$ with $-[\hat{\omega}]^2$ etc, gives the following:

$$\begin{aligned} e^{[\hat{\omega}]\theta} &= I + [\hat{\omega}]\theta + [\hat{\omega}]^2\frac{\theta^2}{2!} + [\hat{\omega}]^3\frac{\theta^3}{3!} + [\hat{\omega}]^4\frac{\theta^4}{4!} + [\hat{\omega}]^5\frac{\theta^4}{5!} \\ &= I + (\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \ldots)[\hat{\omega}] + (\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \ldots)[\hat{\omega}]^2 \end{aligned} \tag{5.16}$$

The Taylor Series expansion for $sin(\theta)$ and $cos(\theta)$ are shown below.

$$sin(\theta) = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \ldots \tag{5.17}$$

$$cos(\theta) = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \ldots \tag{5.18}$$

Substituting the above expansion of the trigonometric functions into equation 5.16 gives the closed-form solution of the matrix exponential $e^{[\hat{\omega}]\theta}$

$$e^{[\hat{\omega}]\theta} = I + sin(\theta)[\hat{\omega}] + (1 - cos(\theta))[\hat{\omega}]^2 \tag{5.19}$$

The formula above is known as the Rodrigues formula for rotations [36].

5.3 Screw Motion

Chasles' Theorem states that every rigid body transformation can be made up of a rotation about an axis and a translation along that same axis. It is an equivalent transformation as turning a screw and is therefore known as a screw transformation [37]. The most common method of representing a screw motion in robotics is by using a homogeneous matrix transformation which is used to describe one coordinate system with respect to another [38].

The axis described above in Chasles' Theorem is known as the screw axis. It is a line along which a translation and about which a rotation is performed with parametrized equation $L(t) = \overline{p} + t\overline{n}$. The equation for the translation of a point by d units along and rotation of a point by θ about the screw axis with parameters $\overline{\omega}$, $\overline{p}$ is shown below [32].

$$\overline{X}' = \overline{P} + e^{[\hat{\omega}]\theta}(\overline{X} - \overline{P}) + d\overline{\hat{\omega}}. \tag{5.20}$$

This can be equated to the rigid-body homogeneous transformation matrix in equation 5.5 as shown below.

$$\overline{R}_{x/y/z} = \begin{bmatrix} e^{\theta[\hat{\omega}]} & (I - e^{\theta[\hat{\omega}]})\overline{P} + d\overline{\hat{\omega}} \\ \overline{0}^T & 1 \end{bmatrix} \tag{5.21}$$

$$H = R_x R_y R_z \tag{5.22}$$

By using the Rodrigues formula (equation 5.19) the above rotation matrix can be solved.

5.4 Forward Kinematics

Forward kinematics is defining the relationship between the joint angles and the end effector configuration such that the joint angles are given and the coordinates of the end effector can be found [22]. The forward kinematics of the hexapod robot is to be found using Denavit-Hartenberg parameters due to the low computational complexity of the solution.

5.4.1 Denavit-Hartenberg Notation

Denavit-Hartenberg (D-H) notation is a system of assigning orthonormal coordinate frames to the joints in a serial, open chain robotic system [39]. It allows for four parameters to represent the system which is beneficial as it reduces number required from six which in turn reduces the computational complexity. By utilizing D-H notation the forward kinematics solution of the robotic system was developed.

Each joint is assigned a coordinate frame by following the steps outlined below [39].

1. Label each joint in the system starting with the base of the robot as a fictitious joint '0', following the serial chain to the end effector labeled joint i.

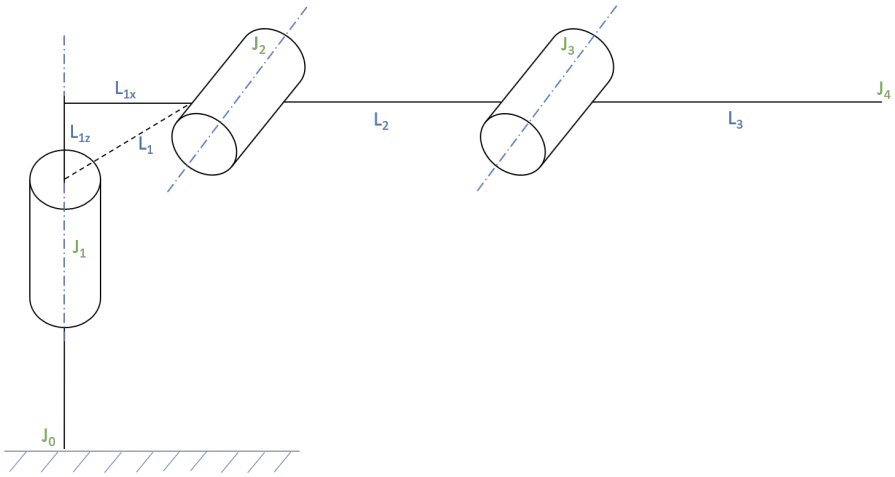


Figure 5.2: D-H Joint Labels

2. Assign a right handed coordinate system to each joint beginning with the base joint. If the joint is rotational in nature, align coordinate frame i such the $z-axis$ points in the direction of the joint axis of joint $i+1$ and the $x-axis$ points along the link towards the next joint. Align the $y-axis$ according to a right-hand coordinate system.

3. Repeat the above steps until you reach the end effector.

4. Assign the end effector frame such that the $z-axis$ points in the direction of the end effector.

5. Align either the $x-axis$ or $y-axis$ such that it is parallel to the previous coordinate frames $x-axis$ or $y-axis$ and align the final axis according to a right hand coordinate frame.

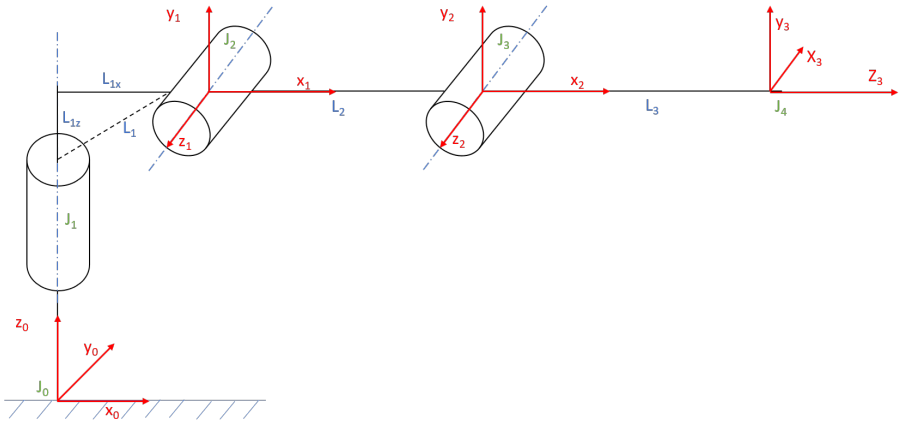


Figure 5.3: D-H Coordinate Frames

6. The joint angle θ_i is the rotation about the $z-axis$ of joint i in a direction which aligns the $x-axis$ of coordinate frame $i-1$ with the $x-axis$ of the previous joint.

7. The twist angle α_i is the rotation about the $x-axis$ of coordinate frame i which will position the $z-axis$ of coordinate frame i parallel to the $z-axis$ of the previous frame.

8. The link length a_i is the length along the $x-axis$ of coordinate frame $i-1$ to $y-z$ plane of frame i.

9. The link offset d_i is the offset along the $z-axis$ of coordinate frame $i-1$ to the $x-z$ plane of frame i.

10. If a joint is rotation then θ is the joint variable and d is constant, if the joint is prismatic then d is the joint variable and θ is constant.

Once each joint's coordinate frame has been assigned and the D-H parameters are found, the homogeneous transformation matrices between each frame can be determined. The D-H parameters for the hexapod are shown in table 5.1 below.

Table 5.1: Hexapod D-H Parameters

Joint	a_i (mm)	d_i (mm)	α_i	θ_i
J_1	53.17	8.0	$90°$	θ_1
J_2	101.88	0.0	0	θ_2
J_3	149.16	0.0	0	θ_3

The rotation matrix from frame i to frame $i-1$ is given by $R_i^{i-1} = U_i V_i$ where U_i is the rotation about the z_{i-1} axis and V_i is the rotation about the x_i axis. This leads to the following matrix for rotation between frames [39].

$$R_i^{i-1} = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i} C_{\alpha_i} & S_{\theta_i} S_{\alpha_i} \\ S_{\theta_i} & C_{\theta_i} C_{\alpha_i} & -C_{\theta_i} S_{\alpha_i} \\ 0 & S_{\alpha_i} & C_{\alpha_i} \end{bmatrix} \tag{5.23}$$

where in the above matrix S_{θ_i} represents $sin(\theta_i)$ and C_{θ_i} represents $cos(\theta_i)$ respectively.

The transformation matrix from frame i to frame $i-1$ is given by $H_i^{i-1} = H_{i'}^{i-1} H_i^{i'}$ where $H_{i'}^{i-1}$ is the screw displacement about the z_i axis through an angle of θ_i and a distance d_i and $H_i^{i'}$ is the screw displacement about the x_i axis through and angle of α_i and a distance a_i. This leads to the following matrix for a transformation between frames.

$$H_i^{i-1} = \begin{bmatrix} C_{\theta_i} & -S_{\theta_i} C_{\alpha_i} & S_{\theta_i} S_{\alpha_i} & C_{\theta_i} a_i \\ S_{\theta_i} & C_{\theta_i} C_{\alpha_i} & -C_{\theta_i} S_{\alpha_i} & S_{\theta_i} a_i \\ 0 & S_{\alpha_i} & C_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.24}$$

By composing multiple frames a transformation from the nth frame to the base frame is found.

$$H_n^0 = H_1^0 H_2^1 H_3^2 ... H_n^{n-1} \tag{5.25}$$

5.4.2 Forward Kinematics Solutions

By applying the Denavit-Hartenberg notation described above to the hexapod the forward kinematics solution of the leg was obtained. Note that in the matrices to follow, S_1 represents $sin(\theta_1)$, S_{23} represents $sin(\theta_2 + \theta_3)$, C_1 represents $cos(\theta_1)$ and C_{23} represents $cos(\theta_2 + \theta_3)$

Transformation from frame 1 to frame 0.

$$H_1^0 = \begin{bmatrix} C_{\theta_1} & -S_{\theta_1}C_{\alpha_1} & S_{\theta_1}S_{\alpha_1} & C_{\theta_1}a_1 \\ S_{\theta_1} & C_{\theta_1}C_{\alpha_1} & -C_{\theta_1}S_{\alpha_1} & S_{\theta_1}a_1 \\ 0 & S_{\alpha_1} & C_{\alpha_1} & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_1 & 0 & S_1 & L_{1_x}C_1 \\ S_1 & 0 & -C_1 & L_{1_x}S_1 \\ 0 & 1 & 0 & L_{1_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.26}$$

Transformation from frame 2 to frame 1.

$$H_2^1 = \begin{bmatrix} C_{\theta_2} & -S_{\theta_2}C_{\alpha_2} & S_{\theta_2}S_{\alpha_2} & C_{\theta_2}a_2 \\ S_{\theta_2} & C_{\theta_2}C_{\alpha_2} & -C_{\theta_2}S_{\alpha_2} & S_{\theta_2}a_2 \\ 0 & S_{\alpha_2} & C_{\alpha_2} & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_2 & -S_2 & 0 & L_2C_2 \\ S_2 & C_2 & 0 & L_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.27}$$

Transformation from frame 3 to frame 2.

$$H_3^2 = \begin{bmatrix} C_{\theta_3} & -S_{\theta_3}C_{\alpha_3} & S_{\theta_3}S_{\alpha_3} & C_{\theta_3}a_3 \\ S_{\theta_3} & C_{\theta_3}C_{\alpha_3} & -C_{\theta_3}S_{\alpha_3} & S_{\theta_3}a_3 \\ 0 & S_{\alpha_3} & C_{\alpha_3} & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_3 & -S_3 & 0 & L_3C_3 \\ S_3 & C_3 & 0 & L_2S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.28}$$

By multiplying each transformation the total forward kinematics solution was found.

$$T_3^0 = H_1^0 H_2^1 H_3^2 \tag{5.29}$$

$$T_3^0 = \begin{bmatrix} C_1C_{23} & -C_1S_{23} & S_1 & C_1(L_{1_x} + L_3C_{23} + L_2C_2) \\ S_1C_{23} & -S_1S_{23} & -C_1 & S_1(L_{1_x} + L_3C_{23} + L_2C_2) \\ S_{23} & C_{23} & 0 & L_{1_z} + L_3S_{23} + L_2S_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.30}$$

5.5 Differential Kinematics

It is necessary to define the relationship between joint velocities and the end effector velocity. This is done by calculating the Jacobian of the forward kinematics solution, named the manipulator Jacobian [34]. The manipulator Jacobian is a $6 \times n$ matrix, where n is the number of links in the manipulator, describing the instantaneous transformation between the n-vector of joint velocities and the 6-vector of end effector velocities.

The manipulator Jacobian also helps us to understand singular configurations [31]. This is a position at which the manipulator loses one or more degrees of freedom.

5.5.1 The Derivative of a Rotation Matrix

Suppose that the rotation matrix $R(\theta)$ exists and the following is true [34]:

$$R(\theta)R(\theta)^T = I \tag{5.31}$$

Deriving both sides of the equation gives:

$$\frac{dR}{d\theta}R(\theta)^T + R(\theta)\frac{dR^T}{d\theta} = 0 \tag{5.32}$$

We define the matrix $S = \frac{dR}{d\theta}R(\theta)^T$. This leads to $S^T = [\frac{dR}{d\theta}R(\theta)^T]^T = R(\theta)\frac{dR^T}{d\theta}$ which satisfies the condition of $S + S^T = 0$ for a skew symmetric matrix.

This shows that the derivative of a rotation matrix is simply $\frac{dR}{d\theta} = SR(\theta)$.

This says that the derivative of a rotation matrix is equivalent to the multiplication by a skew symmetric matrix S.

5.5.2 Angular Velocity

Following on from the above derivation of the derivative of a rotation matrix assuming that $R(t)$ is a rotation matrix that varies continuously with time, the time derivative of this function is $\dot{R}(t) = S(t)R(t)$.

The properties of the skew symmetric matrix S allows $S(t)$ to be presented as $S(\omega(t))$ where $\omega(t)$ is the angular velocity of system.

Therefore the time derivative of $R(t)$ is shown to be $\dot{R}(t) = S(\omega(t))R(t)$ where $\omega(t)$ is the angular velocity. This shows a relationship between angular velocity and the time derivative of the rotation matrix.

5.5.3 Derivation of the Jacobian

The total Jacobian is a concatenation of the angular velocity and linear velocity Jacobians as shown below where J_v is the linear velocity component and J_ω is the angular velocity component.

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{5.33}$$

Angular Velocity

Angular velocities can be added to one another if they are expressed relative to the same coordinate system. Therefore the angular velocity of the end effector relative to the robot base can be determined by summing the angular velocities of each joint expressed in the base frame.

Assuming the frame and joint numbering convention according to Denavit-Hartenberg Notation the angular velocity expressed in frame $i-1$ for link i is $\omega_i^{i-1} = \dot{\theta}_i k$ where k is the unit coordinate vector $(0,0,1)^T$.

This leads to the following overall angular velocity of the end effector.

$$\omega_3^0 = \dot{\theta}_1 k + \dot{\theta}_2 R_1^0 k + \dot{\theta}_3 R_2^0 k \tag{5.34}$$

The expression $R_{i-1}^0 k$ is equivalent to the z-axis of rotation of frame $i-1$. Therefore $R_{i-1}^0 k = z_{i-1}^0$.

This leads to the following angular velocity Jacobian for the hexapod robot.

$$J_\omega = \begin{bmatrix} z_0^0 & z_1^0 & z_2^0 \end{bmatrix} \tag{5.35}$$

Using the homogeneous transformation matrices developed in the forward kinematics of the hexapod these z vectors are determined.

$$z_0^0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad z_1^0 = \begin{bmatrix} S_1 \\ -C_1 \\ 0 \end{bmatrix} \qquad z_2^0 = \begin{bmatrix} S_1 \\ -C_1 \\ 0 \end{bmatrix} \tag{5.36}$$

Linear Velocity

The linear velocity of the end effector is found by using the chain rule for differentiation [34].

$$\dot{o}_n^0 = \sum_{i=1}^{n} \frac{\partial o_n^0}{\partial \theta_i} \dot{\theta}_i \tag{5.37}$$

This leads to the linear velocity Jacobian column of:

$$J_{v_i} = \frac{\partial o_n^0}{\partial \theta_i} \tag{5.38}$$

where o_n^0 represents the position of frame n with respect to frame 0.

From our D-H convention T_n^0 can be expanded as follows.

$$T_n^0 = T_{i-1}^0 T_i^{i-1} T_n^i = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \tag{5.39}$$

$$= \begin{bmatrix} R_{i-1}^0 & o_{i-1}^0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_i^{i-1} & o_i^{i-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_n^i & o_n^i \\ 0 & 1 \end{bmatrix} \tag{5.40}$$

$$= \begin{bmatrix} R_n^0 & R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \\ 0 & 1 \end{bmatrix} \tag{5.41}$$

which gives

$$o_n^0 = R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1} + o_{i-1}^0 \tag{5.42}$$

To solve equation 5.38 for the i-th column of the Jacobian one holds all joints fixed but joint i and actuates it at unit velocity. If only joint i is moved while all other joints are held still then o_n^i and o_{i-1}^0 are constant as well as R_{i-1}^0. Substituting equation 5.42 into equation 5.38 leads to the following.

$$\frac{\partial}{\partial \theta_i} o_n^0 = \frac{\partial}{\partial \theta_i} (R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}) \tag{5.43}$$

$$= o_n^i \frac{\partial}{\partial \theta_i} R_i^0 + R_{i-1}^0 \frac{\partial}{\partial \theta_i} o_i^{i-1} \tag{5.44}$$

The first term in equation 5.44 is solved by using the time derivative of a rotation matrix solution of $\dot{R}(t) = S(\omega(t))R(t)$ where $\omega(t)$ is the z-axis rotation.

$$o_n^i \frac{\partial}{\partial \theta_i} R_i^0 = o_n^i S(z_{i-1}^0) R_i^0 \dot{\theta}_i \tag{5.45}$$

The second term in equation 5.44 is solved as follows.

Substituting in $o_i^{i-1} = (aC_i, aS_i, d_i)^T$ gives the following:

$$R_{i-1}^0 \frac{\partial}{\partial \theta_i} \begin{bmatrix} aC_i \\ aS_i \\ d_i \end{bmatrix} = R_{i-1}^0 \begin{bmatrix} -aS_i \\ aC_i \\ 0 \end{bmatrix} \dot{\theta} \tag{5.46}$$

$$= R_{i-1}^0 S(k\dot{\theta}_i) o_i^{i-1} \tag{5.47}$$

$$= R_{i-1}^0 S(k\dot{\theta}_i)[(R_{i-1}^0)^T R_{i-1}^0] o_i^{i-1} \tag{5.48}$$

$$= S(R_{i-1}^0 k\dot{\theta}_i) R_{i-1}^0 o_i^{i-1} \tag{5.49}$$

$$= \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \tag{5.50}$$

The above manipulation was done using the properties of the skew symmetric matrix S.

Substituting the above into equation 5.44 [34]:

$$\frac{\partial}{\partial \theta_i} o_n^0 = o_n^i S(z_{i-1}^0) R_i^0 \dot{\theta}_i + \dot{\theta}_i S(z_{i-1}^0) R_{i-1}^0 o_i^{i-1} \tag{5.51}$$

$$= \dot{\theta}_i S(z_{i-1}^0)(R_i^0 o_n^i + R_{i-1}^0 o_i^{i-1}) \tag{5.52}$$

$$= \dot{\theta}_i S(z_{i-1}^0)(o_n^0 - o_{i-1}^0) \tag{5.53}$$

$$= \dot{\theta}_i z_{i-1}^0 \times (o_n^0 - o_{i-1}^0) \tag{5.54}$$

Therefore,

$$J_{v_i} = z_{i-1}^0 \times (o_n^0 - o_{i-1}^0) \tag{5.55}$$

Total Jacobian

Combining the above angular and linear velocity Jacobians leads to the final Jacobian solution for revolute joints as shown in equation 5.56 below.

$$J_i = \begin{bmatrix} z_{i-1}^0 \times (o_n^0 - o_{i-1}^0) \\ z_{i-1} \end{bmatrix} \tag{5.56}$$

From the final Jacobian above the end effector velocity can be calculated as follows:

$$\dot{X} = \begin{bmatrix} v_n^0 \\ \omega_n^0 \end{bmatrix} = J\dot{\theta} \tag{5.57}$$

5.5.4 Differential Kinematics Solutions

By applying the methods derived above, the Jacobian for the hexapod was determined.

$$J = \begin{bmatrix} -S_1(L_{1_x} + L_2C_2 + L_3C_{23}) & -C_1(L_2S_2 + L_3S_{23}) & -L_3C_1S_{23} \\ C_1(L_{1_x} + L_2C_2 + L_3C_{23}) & -S_1(L_2S_2 + L_3S_{23}) & -L_3S_1S_{23}) \\ 0 & L_2C_2 + L_3C_{23} & L_3C_{23} \\ 0 & S_1 & S_1 \\ 0 & -C_1 & -C_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.58}$$

5.6 Inverse Kinematics

Inverse kinematics is the process of finding a relationship between the end effector position and the joint variables. It is done in the reverse direction to forward kinematics thus yielding joint variables for any given end effector position.

Inverse kinematics is crucial for robot control however it is complex. A single position of the end effector may be achievable by multiple different joint variable values (angles or prismatic distances), single joint variable values or there may be no solution at all [31].

Numerical methods can be used to solve the inverse kinematic problem, however it will only give a single solution and if there is in fact no solution, the calculation will never converge and the computation will never complete.

Closed form, algebraic solutions are possible and will yield all possible solutions to the problem including the 'no solution' case. This decreases the computational complexity of the kinematics solution and allows for the smart selection of which solution to use based on given criteria. For the case of the hexapod each leg is made up from an anthropomorphic manipulator which consists of only three joints. This allows the inverse kinematics to be found with trigonometry.

Singularities

Singularities occur when the manipulator is in a configuration such that it loses a degree of freedom. For example, if the hexapod leg is in the configuration shown in figure 5.4 below it will not be able to move in the x-direction as it is already fully extended.

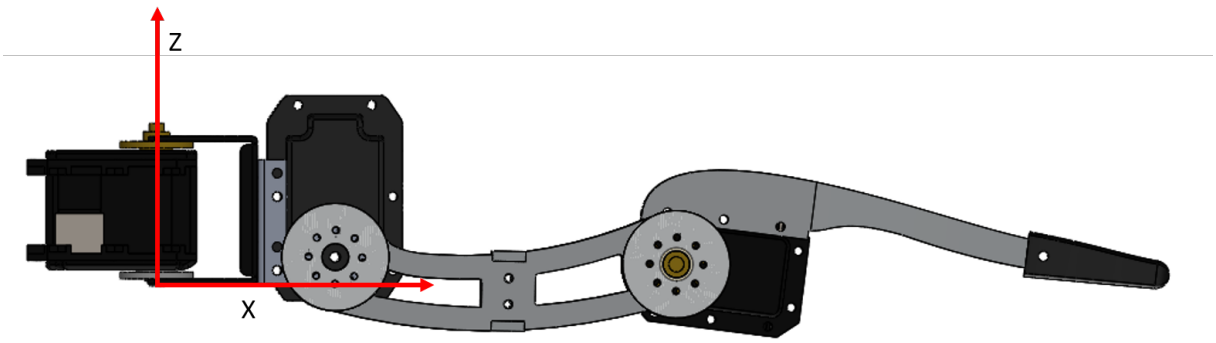


Figure 5.4: Hexapod Robot in Singularity Configuration

5.6.1 Inverse Kinematics Solutions

A model of the hexapod leg can be seen in figure 5.5 below. This was used to determine the inverse kinematics solution of the leg.

The trigonometric function $atan2$ was used instead of $atan$ so that a full range of angles can be calculated. $atan$ outputs an angle from $-90°$ to $90°$ while $atan2$ outputs and angle from $-180°$ to $180°$.

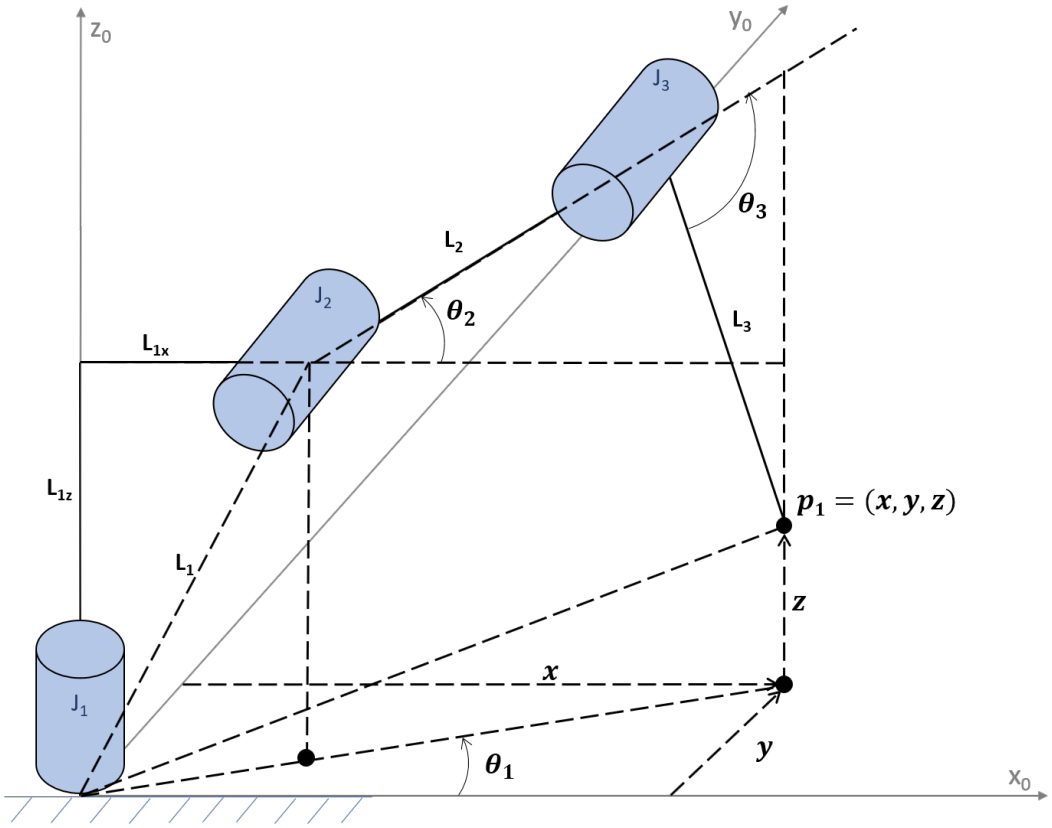


Figure 5.5: Hexapod Robot Leg Inverse Kinematics Model

$$\theta_1 = atan2(y, x) \tag{5.59}$$

$$\theta_2 = atan2(z - L_{1_z}, \sqrt{(x - L_{1_x}C_1)^2 + (y - L_{1_x}S_1)^2}) - atan2(L_3S_3, L_2 + L_3C_3) \tag{5.60}$$

$$\theta_3 = atan2(S_3, C_3) \tag{5.61}$$

where C_3 is found using the rule of cosines [34].

$$C_3 = \frac{(x - L_{1_x}C_1)^2 + (y - L_{1_x}S_1)^2 + (z - L_{1_z})^2 - L_2^2 - L_3^2}{2L_2L_3} \tag{5.62}$$

$$S_3 = \pm\sqrt{1 - C_3^2} \tag{5.63}$$

The variable S_3 is either positive or negative depending on whether the leg is in an 'elbow up' or 'elbow down' position. Elbow up is when the end effector is above the joint J_3 while elbow down is when it is below the joint.

For the hexapod to achieve a walking motion the leg must always be in an elbow down position and therefore S_3 is always negative.

5.6.2 Inverse Differential Kinematics

The inverse differential kinematics is found by using the inverse of the Jacobian. This is trivial if the Jacobian is square and non-singular [34].

$$\dot{X} = J\dot{\theta} \tag{5.64}$$

$$J^{-1}\dot{X} = J^{-1}J\dot{\theta} \tag{5.65}$$

$$\therefore \dot{\theta} = J^{-1}\dot{X} \tag{5.66}$$

If the robotic system does not have exactly six joints then the Jacobian will not be square and therefore it cannot be inverted. In order to calculate the inverse differential kinematics of a system where the Jacobian cannot be inverted one must calculate the Moore-Penrose pseudoinverse Jacobian, J^{+}.

Assuming that the robot is not at a singularity, for the Jacobian $J \in \mathbb{R}^{m\times n}$, where m is the number of columns and n is the number of rows, if $n > m$ the inverse Jacobian is calculated using the left pseudoinverse while if $n < m$ it is calculated using the right pseudoinverse [31].

$$J^{+} = (J^{T}J)^{-1}J^{T} \qquad \textbf{Left} \text{ Pseudoinverse of the Manipulator Jacobian} \tag{5.67}$$

$$J^{+} = J^{T}(J^{T}J)^{-1} \qquad \textbf{Right} \text{ Pseudoinverse of the Manipulator Jacobian} \tag{5.68}$$

5.6.3 Inverse Differential Kinematics Solutions

$$J^T = \begin{bmatrix} -S_1(L_{1_x} + L_2C_2 + L_3C_{23}) & C_1(L_{1_x} + L_2C_2 + L_3C_{23}) & 0 & 0 & 0 & 1 \\ -C_1(L_2S_2 + L_3S_{23}) & -S_1(L_2S_2 + L_3S_{23}) & L_2C_2 + L_3C_{23} & S_1 & -C_1 & 0 \\ -L_3C_1S_{23} & -L_3S_1S_{23} & L_3C_{23} & S_1 & -C_1 & 0 \end{bmatrix} \tag{5.69}$$

$$J^+ = (J^T J)^{-1} J^T \tag{5.70}$$

$$\dot{\theta} = J^+ \dot{X} \tag{5.71}$$

where $\dot{X}$ is a vector describing the linear and angular velocity of the foot $\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix}$

Note that the matrix J^+ is too large to be displayed neatly and for this reason it has not been shown, however it can be calculated using equation 5.70.

The angular velocity of the foot of the hexapod in relation to the motor angles of the foot are related through equation 5.34.

The angular velocity around the $x-axis$ is equal to $(\dot{\theta}_2+\dot{\theta}_3)S_1$, around the $y-axis$ is equal to $-(\dot{\theta}_2+\dot{\theta}_3)C_1$ and around the $z-axis$ is equal to $\dot{\theta}_1$. This leads to the following equations after rearranging in terms of $\dot{\theta}$.

$$\dot{\theta}_1 = \frac{\dot{y}C_1 - \dot{x}S_1}{L_{1x} + L_3C_{23} + L_2C_2} \tag{5.72}$$

$$\dot{\theta}_2 = \frac{1}{L_2}\left[\dot{z}C_2 - \dot{x}C_1S_2 - \dot{y}S_1S_2 + \frac{C_3}{S_3}\left(\dot{z}S_2 + \dot{x}C_1C_2 + \dot{y}C_2S_1\right)\right] \tag{5.73}$$

$$\dot{\theta}_3 = -\frac{1}{L_2}\left[\dot{z}C_2 - \dot{x}C_1S_2 - \dot{y}S_1S_2 + \frac{L_2 + L_3C_3}{L_3S_3}\left(\dot{z}S_2 + \dot{x}C_1C_2 + \dot{y}C_2S_1\right)\right] \tag{5.74}$$

The equation below is used to calculated the angular acceleration of each joint in relation to the angular acceleration of the end effector. The hexapod motors have built in controllers and as such this acceleration value is not needed to control the hexapod and is therefore not implemented in the hexapod control system.

$$\ddot{\theta} = J^+(\ddot{X} - \frac{dJ}{dt}\dot{\theta}) \tag{5.75}$$

5.7 Implementing Kinematics

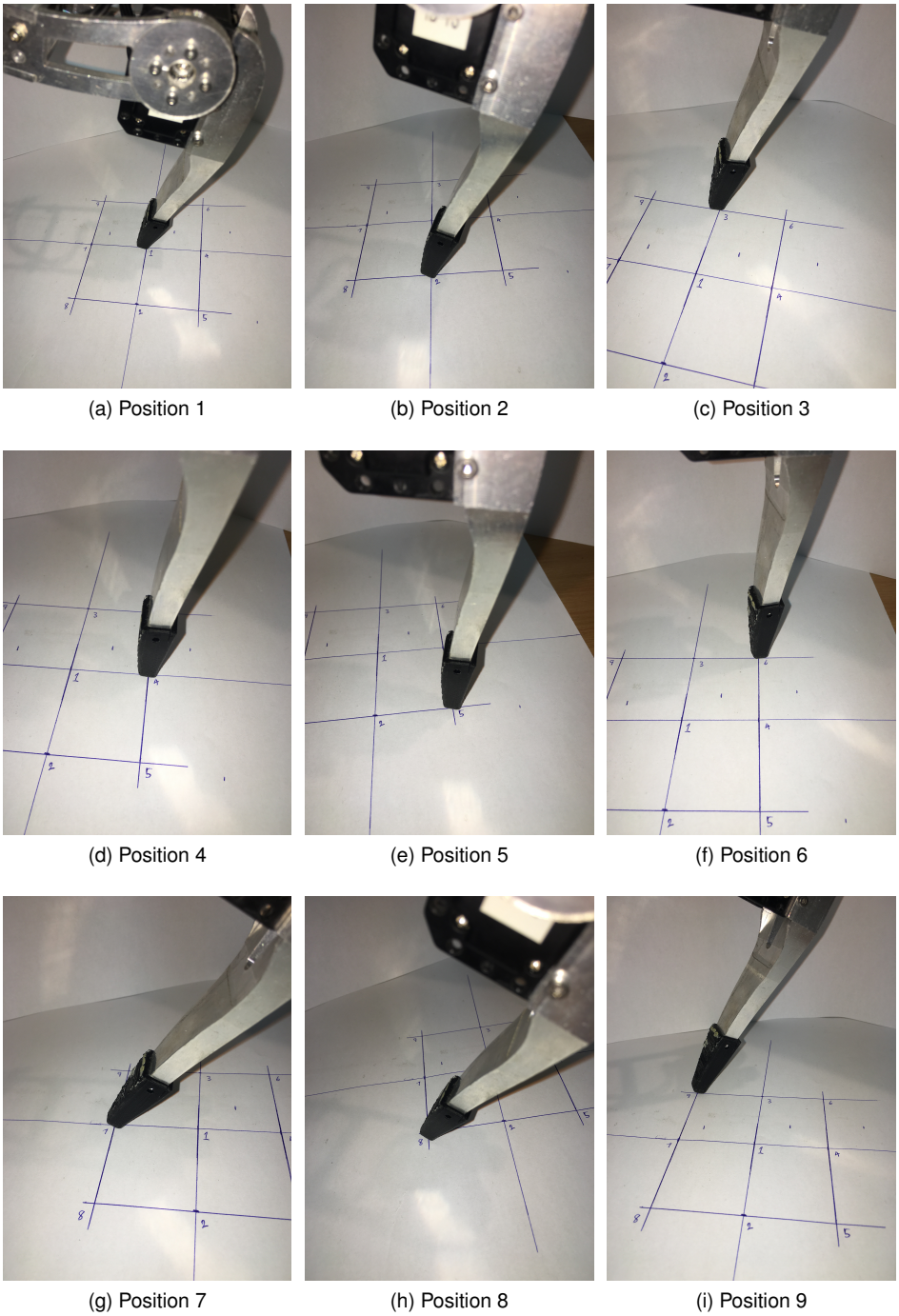
The inverse and differential kinematics determined previously was implemented in C++ on the micro-controller of the hexapod. When a foot XYZ position is given, the inverse kinematics is run to determine the joint angles required to achieve the commanded foot position. The same process applies to the speed of the foot. When $\dot{X}, \dot{Y}, \dot{Z}$ speeds are given, the inverse differential kinematics is run to determine the rotational speed of each joint required to achieve the commanded foot velocity.

5.8 Kinematics Testing and Verification

Testing was performed to verify that the position and speed control, implemented on the micro-controller, responded accurately according to the mathematics detailed previously.

5.8.1 Position Control

To test the accuracy of the position control of the legs of the hexapod, the leg was commanded to move to various different points which were marked on paper at known locations. The accuracy of the position at each point as well as the repeatability for each point was recorded. Images of the test in progress can be seen in figure 5.6 below.



(a) Position 1 (b) Position 2 (c) Position 3

(d) Position 4 (e) Position 5 (f) Position 6

(g) Position 7 (h) Position 8 (i) Position 9

Figure 5.6: Leg Kinematics Test

The previous images show the first iteration of the test. The test was repeated and marks were made on the test sheet at each location the leg moved to. This can be seen in figure 5.7 below.

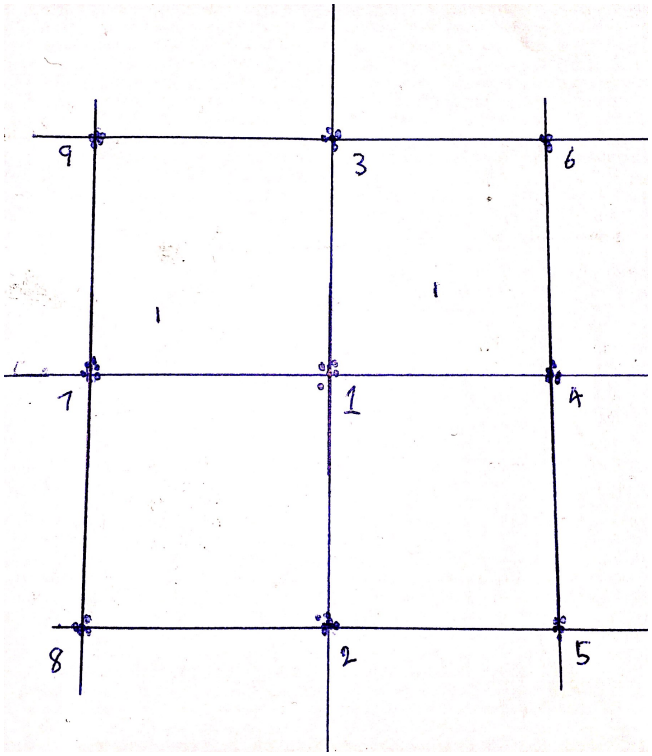


Figure 5.7: Leg Kinematics Test Results

From the results in the figure above, the accuracy and repeatability of the leg motion was determined. This was done by calculating the mean error and standard deviation at each point and averaging these values over all nine points. Note that the error was calculated as the radial distance from the commanded position to the actual position.

This lead to mean error of 0.99mm and a standard deviation of 0.58mm.

The vertical position was also tested by moving the foot to a known vertical height and measuring this distance as in the figure below.

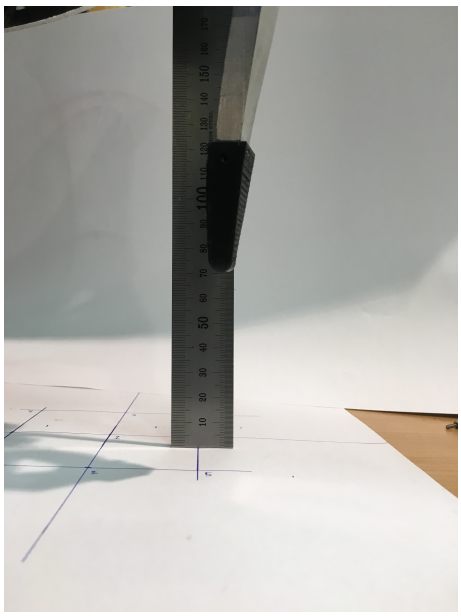


Figure 5.8: Leg Kinematics Vertical Height Test

After multiple tests at different heights were performed, the average error was calculated to be 1.70mm with a standard deviation of 0.68mm.

5.8.2 Speed Control

To accurately test the speed control of the hexapod's leg is a complex problem. One such method, is to simply move the leg through a path with a known length and record the time it takes from its starting point to its finishing point. From this time, and the path length, the speed of the foot tip can be calculated.

This test was repeated for multiple different paths, however for simplicity of testing and calculations only straight line paths were used. The speed of the leg was set using the inverse differential kinematics calculations and this speed was then verified through this testing.

The table below shows the recorded data for five speed tests over three different path lengths. Note that path 1 was 160mm in the XYZ direction, path 2 was 80mm in the X direction and path 3 was 30mm in the Y direction. This was done so that the data is inclusive of the speeds of all motors.

Table 5.2: Hexapod Leg Speed Test Results

Path 1 (s)	Path 2 (s)	Path 3 (s)	Set Speed (mm/s)	Calculated Speed (mm/s)
0.31	0.17	N/A	500.0	493.35
1.42	0.69	0.38	100.0	101.43
3.22	1.52	0.57	50.0	51.65
7.83	3.41	1.76	20.0	20.31
17.45	6.19	3.03	10.0	10.66

From the data in the table above the average error on the speed control was calculated to be 2.36mm/s with a standard deviation of 9.03mm/s.

5.9 Discussion of Kinematics

The mathematical modelling of the leg of the hexapod robot lead to the development of the forward kinematics, differential kinematics, inverse kinematics and the inverse differential kinematics solutions. These equations were successfully implemented in C++ and were able to accurately control both the position and speed of the foot of the hexapod with a high degree of repeatability.

Position testing was done by moving the leg to multiple known points and recording the actual position of the foot and comparing this to the commanded position of the foot. Sources of error in the position and speed control include rounding errors when converting between degrees and radians and rad/s and rpm. The motors have a large amount of jitter when there is no load present caused by the dynamixel internal controller. Due to this, during the position test the foot would often shake slightly over a position point making it difficult to mark its position accurately. This is another source for errors in the position control and in the tests performed.

The speed control test was performed using a stopwatch and measuring the time taken for the leg to traverse a known path. This time was then used to calculated the actual speed and this was compared to the commanded speed to determine the error in speed control. Jitter did not affect this test however human error with starting and stopping the stopwatch may have produced inaccuracies in the data. At the high speed of 500mm/s there was little time between starting and stopping the stopwatch and therefore this data was particularly inaccurate. Path 3 at the speed of 500mm/s was excluded from the data as it was too fast to be able to measure accurately and thus skewed the data.

Motor backlash was noticed on both tests. Backlash refers to when the output shaft of the motor gearbox moves without the input shaft turning [40]. This is caused by tolerances in the meshing of the teeth of the gearbox inside the servo motors. Although the backlash was small it can affect the repeatability of the testing as different amounts of backlash can occur in one test when compared to another due to factors such as the motor position at the start of a test, the movement speed and the load during the test.

6 Leg Dynamics

The dynamics of the robot define the relationship between the torques exerted by the motors on the joints of the robot and the forces exerted at the end effector. This is important when trying to specify a gripping force at the end effector or when trying to determine whether or not the robot has collided with an obstacle. Using torque feedback from each motor one can determine if the torque spikes (for example on impact with an object) and, using the dynamics of the system, calculate the force exerted on the foot of each leg [41].

To develop the dynamic model the Euler-Lagrange equations are used. The difference between the kinetic and potential energy is calculated as show below. This is known as the Lagrangian, $\mathcal{L}$.

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{6.1}$$

where $\mathcal{K}$ is the kinetic energy and $\mathcal{P}$ is the potential energy of the system.

From the Lagrangian above, the equations of motion of the system are defined as shown in equation 6.2 below.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \tag{6.2}$$

This equation is the general form of the Euler-Lagrange equation of motion. The variable q represents the joints variables of the robotic system. For the case of the hexapod, this is the Denavit-Hartenberg joint variables θ_i, while τ_i is the torques at each joint [41].

6.1 Kinetic Energy

The kinetic energy of a 2D object is defined as $\frac{1}{2}mv^2$ where m is the mass of the object and v is the velocity. This accounts for linear motion only. The kinetic energy of a rigid body in three dimensional space must account for rotational kinetic energy as well as linear kinetic energy. This is done by using equation 6.3 below.

$$\mathcal{K} = \frac{1}{2}[mv^T v + \omega^T \mathcal{I}\omega] \tag{6.3}$$

where v is the linear velocity vector, ω is the angular velocity vector and $\mathcal{I}$ is the Inertia Tensor [41].

6.1.1 The Inertia Tensor

The linear and angular velocity vectors are expressed in the inertial or 'global' coordinate system. It is therefore important that the inertia tensor is also expressed in the inertial system. An inertia tensor in the coordinate system of the rigid body, or body coordinate system, can be transformed into the inertial frame through the following manipulation.

$$\mathcal{I} = RIR^T \tag{6.4}$$

where R is the rotation matrix from the body frame to the inertial frame and I is the inertia tensor in the body frame.

Determining the inertia tensor in this way is required because the inertia tensor in the inertial frame is not constant while the inertia tensor in the body frame is constant. The inertia tensor in the body frame can be determined from first principles using volumetric integrals over the space occupied by the rigid body link [41].

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{6.5}$$

where

$$I_{xx} = \iiint (y^2 + z^2)\rho(x, y, z)dxdydz \tag{6.6}$$

$$I_{yy} = \iiint (x^2 + z^2)\rho(x, y, z)dxdydz \tag{6.7}$$

$$I_{zz} = \iiint (x^2 + y^2)\rho(x, y, z)dxdydz \tag{6.8}$$

$$I_{xy} = I_{yx} = -\iiint xy\rho(x, y, z)dxdydz \tag{6.9}$$

$$I_{xz} = I_{zx} = -\iiint xz\rho(x, y, z)dxdydz \tag{6.10}$$

$$I_{yz} = I_{zy} = -\iiint yz\rho(x, y, z)dxdydz \tag{6.11}$$

For complicated rigid body shapes the above calculations become increasingly difficult. If an accurate CAD model of the system is developed one can determine the body inertia matrix of each rigid body link using Solidworks [42].

6.1.2 Jacobian Kinetic Energy

In the previous section it was shown that the linear and angular velocities of a point on a rigid body can be expressed in terms of the Jacobian at that point. Therefore the linear and angular velocity of a point is:

$$v_i = J_{v_i}(q)\dot{q} \tag{6.12}$$

$$\omega_i = J_{\omega_i}(q)\dot{q} \tag{6.13}$$

where q is a general variable that is replaced with θ for the hexapod.

Following on from equation 6.3 and including equation 6.4 leads to the following final equation for the kinetic energy of a general robot system with n links.

$$\mathcal{K} = \frac{1}{2}\dot{q}^T \sum_{i=1}^{n} \Big[m_i J_{v_i}^T(q) J_{v_i}(q) + J_{\omega_i}^T(q) R_i(q) I_i R_i^T(q) J_{\omega_i}(q) \Big] \dot{q} \tag{6.14}$$

where m_i is the mass of link i, I_i is the inertia matrix of link i in the body frame calculated at the center of mass of the link and R_i is a rotation matrix from link i to the inertial frame.

This can be put into the form

$$K = \frac{1}{2}\dot{q}^T M(q)\dot{q} \tag{6.15}$$

where $M(q)$ is known as the inertia matrix [41].

6.2 Potential Energy

For a rigid body robotic system the only potential energy source is gravity. Assuming that the mass of the entire rigid body link is acting at the center of mass of the link and that the center of mass of the link is at the geometric center of the link, the potential energy of the link can be calculated as show below.

$$\mathcal{P} = g^T r_{ci} m_i \tag{6.16}$$

where g is the gravity vector pointing in the direction of gravity in the inertial frame, r_{ci} is a vector giving the coordinates of the center of mass of the link and m_i is the mass of the link [41].

In the above equations it is assumed that the links do not deform. In real world applications this is not the case, however the potential energy from the elastic deformation of the aluminium links under the loads the hexapod experiences is assumed to be negligible when compared to the potential energy due to gravity. This can be proven with an elastic deformation calculation for each link.

The total potential energy of the robotic system is the sum of the potential energy of each link.

$$\mathcal{P} = \sum_{i=1}^{n} g^T r_{ci} m_i \tag{6.17}$$

6.3 Equations of Motion

The Euler-Lagrange equations can be further manipulated for the case of robotic systems. This manipulation can only be done if the following conditions are satisfied:

- The kinetic energy is a quadratic function of the joint variable $\dot{q}$ in the form of equation 6.15.
- The potential energy is independent of $\dot{q}$.

The Euler-Lagrange equations are then derived as follows [31], [41].

$$\mathcal{L} = \mathcal{K} - \mathcal{P} = \frac{1}{2}\dot{q}^T M(q)\dot{q} - \mathcal{P}(q) \tag{6.18}$$

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} m_{ij}(q)\dot{q}_i\dot{q}_j - \mathcal{P}(q) \tag{6.19}$$

substituting equation 6.19 above into equation 6.2 the following solution is found [43].

$$\tau_i = \sum_{j=1}^{n} m_{ij}\ddot{q}_j + \sum_{j=1}^{n}\sum_{k=1}^{n}\Gamma_{ijk}(q)\dot{q}_j\dot{q}_k + \frac{\partial \mathcal{P}}{\partial q_i} \tag{6.20}$$

where $\Gamma_{ijk}(q)$ is known as the Christoffel symbols of the first kind and is defined as follows [43].

$$\Gamma_{ijk}(q) = \frac{1}{2}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i}\right) \tag{6.21}$$

The Christoffel symbols defined above represent the Coriolis and centripetal terms in the dynamics equation. It is seen from equation 6.21 that the Christoffel symbols are derived from the inertia matrix M.

Rewriting equation 6.20 in matrix form leads to the common robotics dynamics equation:

$$Q = M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) \tag{6.22}$$

where Q is the control input torque [44], M is the inertia matrix, C is a matrix of Christoffel symbols defined as $C_{ij}(q,\dot{q}) = \sum_{k=1}^{n}\Gamma_{ijk}(q)\dot{q}_k$ and $G_i(q) = \frac{\partial P}{\partial q_i}$.

6.4 Leg Dynamics Solutions

Using the methods described above the dynamic model of the hexapod's leg was developed. The inertia tensors I_1, I_2 and I_3 are those for links 1, 2 and 3 respectively and were determined using Solidworks. In the equations below the lengths with the notation $L_{...c}$ are lengths to the center of mass of the link.

6.4.1 Mass Inertia Matrix

To find the mass inertia matrix one must start by finding the kinetic energy of the system. This includes the linear and angular kinetic energies. These are determined using equation 6.14. The Jacobian for the center of gravity of each link was determined as well as each link's inertia tensor and rotation matrix.

$$M(1,1) = I_1 + I_2 + I_3 + \frac{L_{2c}^2}{2}\Big[m_2 + m_3 + cos(2\theta_2)(m_2 + m_3)\Big] + L_{3c}^2 m_3\big(1 + cos(2\theta_2 + 2\theta_3)\big) \tag{6.23}$$

$$M(1,2) = 0 \tag{6.24}$$

$$M(1,3) = 0 \tag{6.25}$$

$$M(2,1) = 0 \tag{6.26}$$

$$M(2,2) = L_{2c}^2(m_2 + m_3) + L_{3c}m_3\big(L_{3c} + 2L_{2c}cos(\theta_3)\big) \tag{6.27}$$

$$M(2,3) = L_{3c}m_3\big(L_{3c} + L_{2c}cos(\theta_3)\big) \tag{6.28}$$

$$M(3,1) = 0 \tag{6.29}$$

$$M(3,2) = L_{3c}m_3\big(L_{3c} + L_{2c}cos(\theta_3)\big) \tag{6.30}$$

$$M(3,3) = {L_{3c}}^2 m_3 \tag{6.31}$$

6.4.2 Christoffel Symbols

From the inertia matrix M derived above, the matrix C can be derived as follows.

$$\Gamma_{ijk}(q) = \frac{1}{2}\left(\frac{\partial m_{ij}}{\partial q_k} + \frac{\partial m_{ik}}{\partial q_j} - \frac{\partial m_{jk}}{\partial q_i}\right) \tag{6.32}$$

$$C_{ij}(q,\dot{q}) = \sum_{k=1}^{3} \Gamma_{ijk}(q)\dot{q}_k \tag{6.33}$$

Using MATLab [45] the above equation was used to determine the matrix C.

$$\begin{aligned} C(1,1) = -\frac{\dot{\theta}_3}{2}\Big[& {L_{3c}}^2 m_3 sin(2\theta_2 + 2\theta_3) + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) + L_{2c}L_{3c}m_3 sin(\theta_3) \\ & + L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] - \frac{\dot{\theta}_2}{2}\Big[{L_{2c}}^2 m_2 sin(2\theta_2) \\ & + {L_{2c}}^2 m_3 sin(2\theta_2) + {L_{3c}}^2 m_3 sin(2\theta_2 + 2\theta_3) + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) + \\ & 2L_{2c}L_{1xc}m_2 sin(\theta_2) + 2L_{2c}L_{1xc}m_3 sin(\theta_2) + 2L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] \end{aligned} \tag{6.34}$$

$$\begin{aligned} C(1,2) = -\frac{\dot{\theta}_1}{2}\Big[& 2{L_{2c}}^2 m_2 sin(2\theta_2) + 2{L_{2c}}^2 m_3 sin(2\theta_2) + {L_{3c}}^2 m_3 sin(2\theta_2 + 2theta_3) \\ & + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) + 2L_{2c}L_{1xc}m_2 sin(\theta_2) + 2L_{2c}L_{1xc}m_3 sin(\theta_2) \\ & + 2L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] \end{aligned} \tag{6.35}$$

$$\begin{aligned} C(1,3) = -\frac{\dot{\theta}_1}{2}\Big[& {L_{3c}}^2 m_3 sin(2\theta_2 + 2\theta_3) + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) \\ & + L_{2c}L_{3c}m_3 sin(\theta_3) + L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] \end{aligned} \tag{6.36}$$

$$\begin{aligned} C(2,1) = \frac{\dot{\theta}_1}{2}\Big[& {L_{2c}}^2 m_2 sin(2\theta_2) + {L_{2c}}^2 m_3 sin(2\theta_2) + {L_{3c}}^2 m_3 sin(2\theta_2 + 2\theta_3) \\ & + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) + 2L_{2c}L_{1xc}m_2 sin(\theta_2) + 2L_{2c}L_{1xc}m_3 sin(\theta_2) \\ & + 2L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] \end{aligned} \tag{6.37}$$

$$C(2,2) = -L_{2c}L_{3c}m_3\dot{\theta}_3 sin(\theta_3) \tag{6.38}$$

$$C(2,3) = -L_{2c}L_{3c}m_3\dot{\theta}_2 sin(\theta_3) - L_{2c}L_{3c}m_3\dot{\theta}_3 sin(\theta_3) \tag{6.39}$$

$$C(3,1) = \frac{\dot{\theta}_1}{2}\Big[{L_{3c}}^2 m_3 sin(2\theta_2 + 2\theta_3) + 2L_{3c}L_{1xc}m_3 sin(\theta_2 + \theta_3) + L_{2c}L_{3c}m_3 sin(\theta_3) + L_{2c}L_{3c}m_3 sin(2\theta_2 + \theta_3)\Big] \tag{6.40}$$

$$C(3,2) = L_{2c}L_{3c}m_3\dot{\theta}_2 sin(\theta_3) \tag{6.41}$$

$$C(3,3) = 0 \tag{6.42}$$

6.4.3 Gravity Vector

The gravity vector G is determined from the potential energy of the system. This is calculated using the common potential energy formula $P = mgh$ for the center of gravity of each link.

$$G(1,1) = 0 \tag{6.43}$$

$$G(2,1) = L_{2c}gm_2 cos(\theta_2) + gm_3(L_{3c}cos(\theta_2 + \theta_3) + L_2 cos(\theta_2)) \tag{6.44}$$

$$G(3,1) = L_{3c}gm_3 cos(\theta_2 + \theta_3) \tag{6.45}$$

6.4.4 Equation of motion

Using matrices calculated above and the equation below, the total equation of motion for a single leg of the hexapod was determined.

$$\tau - J^T F_c = M(\theta)\ddot{\theta} + C(\theta, \dot{\theta}) + G(\theta) \tag{6.46}$$

where F_c is the external contact force (ground force) [46]

6.5 Discussion of Results

The dynamics of a single leg of the hexapod robot was successfully determined using the Euler-Lagrange equation for robotics. The potential and kinetic energy of each leg link was found using the mass of each link and the Jacobian of its center of mass. From this the mass inertia matrix was found. Using the mass inertia matrix the Christoffel Symbols were determined and finally the gravity vector was calculated.

Using equation 6.46 which includes the friction forces on the foot, the total equation of motion for a leg of the hexapod was found. The matrices M, C, and G are compared to those of [44] and [30] validating them.

Note that the final dynamics solution was not implemented on the hexapod. This was due to the slow read time of the Dynamixel motors. In order to use the dynamics equations effectively the torque value of each motor needs to be known. The Dynamixel RX series motors do not have a 'sync read' function and as such reading the torque value of all motors takes over 1 second (found experimentally); during which no movement commands can be sent to the motors. This leads to problems with the walking control.

In order to use the dynamics equations the motor torque values must be constantly available. Using these values and equation 6.46 in conjunction with motor positions, speeds and accelerations the foot contact force can be found. This contact force can then be used to determine a gripping force for the legs or used to determine which legs of the hexapod are in contact with the ground.

7 Body Kinematics

The kinematics of the leg developed previously holds only for a local coordinate frame at joint 1 of each leg. This is useful for distributed intelligence, however a global coordinate system must be defined in order to control all of the hexapod's legs uniformly from a central micro-controller.

This is done by using body kinematics to translate the local leg coordinate system into a global hexapod coordinate system. The coordinate systems of each leg and the global, central coordinate system of the hexapod can be seen in figure 7.1 below.

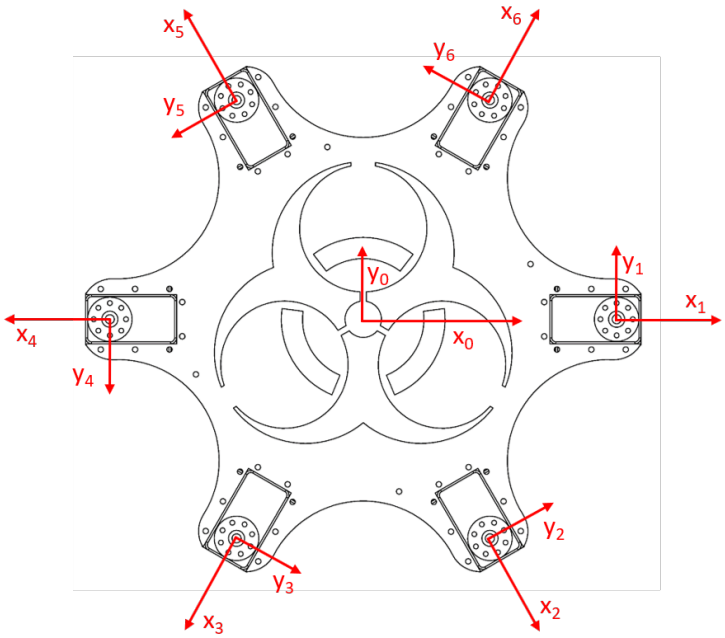


Figure 7.1: Hexapod Robot Coordinate Systems

As seen in the figure above each leg's coordinate system is to be rotated by increments of $60°$ to align with the global coordinate system. It is then translated in the x and y directions to account for the offset to the center of the hexapod. The $z - axis$ of all coordinate systems are already aligned and as such no z adjustments are required.

The transformations depend on the clockwise leg rotation relative to leg 1, α, and the body radius of the hexapod β.

7.1 Forward Kinematics Transformation

The forward kinematics developed previously solves the XYZ position of the leg in the leg's coordinate system given the three θ values of the joints. This XYZ solution must be transformed from the leg's coordinate system into the global hexapod body coordinate system.

$$X_G = Y_L sin(\alpha) + (X_L + \beta)cos(\alpha) \tag{7.1}$$

$$Y_G = Y_L cos(\alpha) - (X_L + \beta)sin(\alpha) \tag{7.2}$$

where α is the leg rotation relative to leg 1, β is the hexapod body radius, X_G and Y_G are the global hexapod coordinates and X_L and Y_L are the local leg coordinates.

7.2 Inverse Kinematics Transformation

The inverse kinematics developed previously solves the θ angles of the leg joints given the XYZ position of the foot in the leg's coordinate systems. This XYZ position must be transformed from the hexapod's global coordinate system into the leg's local coordinate system before the inverse kinematics can be calculated.

$$X_L = X_G cos(\alpha) - Y_G sin(\alpha) - \beta \tag{7.3}$$

$$Y_L = X_G sin(\alpha) + Y_G cos(\alpha) \tag{7.4}$$

7.3 Discussion of Results

The body kinematic equations derived above successfully translate the local leg coordinates into a global body coordinate system and vice versa. These equations are used to provide the user with a global coordinate system specifying points for each leg to move to with reference to the center of the body of the hexapod robot.

This allows for simpler control of the legs as each leg's location is relative to the same, global coordinate system.

8 Leg Motion Planning

Leg motion planning has been separated into two sections, namely foot trajectory planning and foot position planning. Foot trajectory planning consists of planning the path between two points during the swing and support phases of the leg. Foot position planning is the process of determining where the hexapod needs to place each foot to achieve a desired motion and maintain stability.

Foot position planning must happen first as the points that this algorithm determines are used in the foot trajectory planning algorithms.

8.1 Foot Position Planning

The crab angle of the robot is defined as the angle between the direction of the motion of the robot and the direction of the positive x-axis of the robot [47]. This can be seen in the image below. The angle α represents the crab angle of the hexapod.

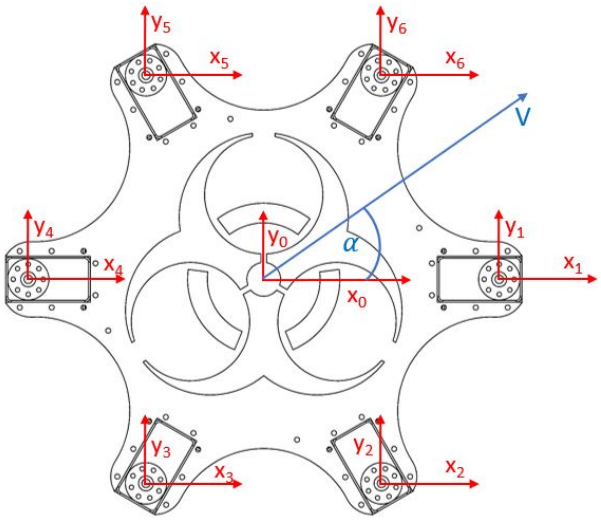


Figure 8.1: Hexapod Coordinate System with Crab Angle

This angle defines the direction in which each leg must move during the support phase in order to propel the hexapod in that direction.

The foot position planning takes in various variables which define the foot motion. These variables include the radial distance from the center of the hexapod to the center of the stride line, the stride length, the step height, the direction of motion (crab angle) and the pitch and roll of the hexapod which indicates the slope on which it is walking. A depiction of the hexapod moving forward, i.e. with a crab angle of zero, showing the strides of each leg to achieve this is shown below.

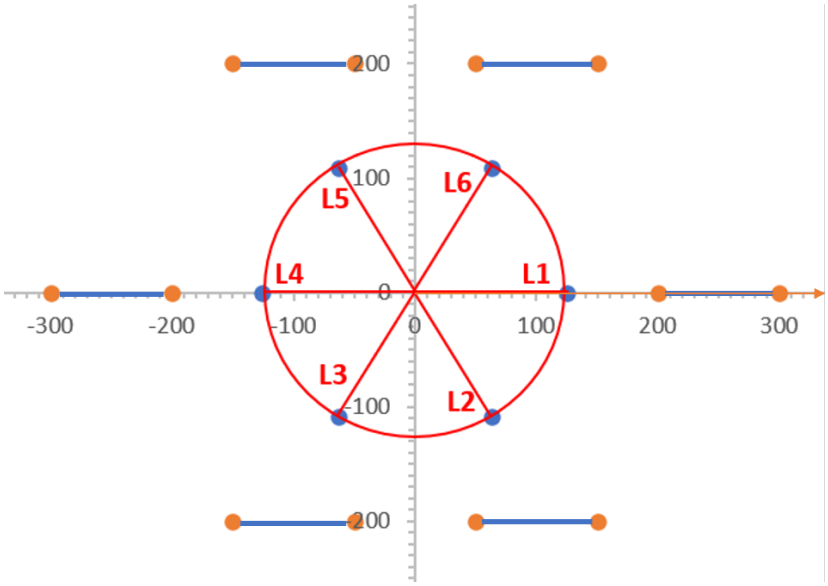


Figure 8.2: Hexapod Leg Strides

The foot position planning algorithm calculates the starting point, mid point and end point for the trajectory of each foot which the path planning algorithm then uses to generate an array of points representing the full path of the foot.

The foot position planning algorithm starts by calculating the start and end height of the foot path from the pitch and roll of the hexapod. These heights are in comparison to the mid point of the stride which is the zero reference point.

$$\Delta h_{start} = \frac{S_s}{2.0}(tan(\theta)cos(\alpha) + tan(\phi)sin(\alpha)) \tag{8.1}$$

$$\Delta h_{end} = -\frac{S_s}{2.0}(tan(\theta)cos(\alpha) + tan(\phi)sin(\alpha)) \tag{8.2}$$

where θ is the pitch of the hexapod, ϕ is roll of the hexapod, α is the crab angle or direction of motion and S_s is the stride length.

Using the above start and end point height changes, the full start, mid and end points are calculated as follows.

Start Point

$$x = R_d cos(R_L) + (\frac{S_s}{2.0})cos(\alpha) \tag{8.3}$$

$$y = -(R_d sin(R_L) + (\frac{S_s}{2.0})sin(\alpha)) \tag{8.4}$$

$$z = -H + \Delta h_{start} \tag{8.5}$$

Mid Point

$$x = R_d cos(R_L) \tag{8.6}$$

$$y = -R_d sin(R_L) \tag{8.7}$$

$$z = -H + S_h \tag{8.8}$$

End Point

$$x = R_d cos(R_L) - (\frac{S_s}{2.0})cos(\alpha) \tag{8.9}$$

$$y = -(R_d sin(R_L) - (\frac{S_s}{2.0})sin(\alpha)) \tag{8.10}$$

$$z = -H + \Delta h_{end} \tag{8.11}$$

where R_L is the rotation of the leg relative to leg one, R_d is the radial distance from the center of the hexapod to the center of the stride of the leg, H is the body height of the hexapod and S_h is the step height of the leg.

8.2 Foot Trajectory Planning

The foot trajectory planning is done on a point to point basis. That is, the foot position planner determines where the foot must be placed and the foot trajectory planning plots a path between these points that satisfies initial and final conditions such as via points, velocity and acceleration.

In the case of the hexapod robot, the path planning technique used is that of via points. This is a method where start and end points are defined as well as any number of points in between the start and end points in order to achieve a desired smooth path. This allows for setting the step height and shape of the swing phase. This is beneficial as it allows for the changing of the foot trajectory based on obstacles in the hexapod's environment.

Polynomial functions can be used to produce a smooth path between points [34]. The initial and final configurations, q_i and q_f respectively, are specified as well as any desired via points, $q_{1,2,3...}$. In order to generate the path, various degrees of polynomials can be used. As the degree of the polynomial increases the smoothness of the path increases and so does the complexity of the calculation.

By specifying the initial position, via positions, final position and velocities a polynomial can be used to generate the path. A linear path between the points is first plotted as a baseline to show how the smoothness improves as the degree of the polynomial increases.

8.2.1 Polynomial Path Generation

The following points were used in the plots below:

$$q_i = 0.2 \qquad \dot{q}_i = 0 \tag{8.12}$$

$$q_1 = 0.25 \qquad \dot{q}_1 = 0 \tag{8.13}$$

$$q_f = 0.3 \qquad \dot{q}_f = 0 \tag{8.14}$$

Note that the plots below are made with two separate polynomial functions to allow the path to go through the via point. One polynomial from q_i to q_1 and another polynomial from q_1 to q_f.

All polynomial coefficients were solved using MATLAB.

Linear Path

The equation below represents the linear trajectory generation (a polynomial of degree one).

$$q(t) = a_0 + a_1 \times t \tag{8.15}$$

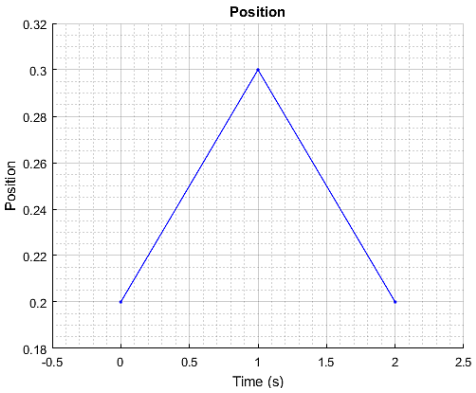
$$\dot{q}(t) = a_1 \tag{8.16}$$

$$\ddot{q}(t) = 0 \tag{8.17}$$

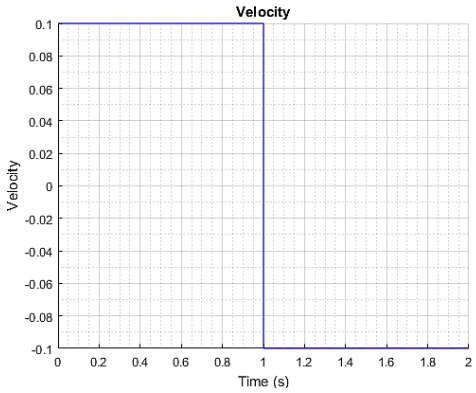
The variables a_0 and a_1 are solved using the initial conditions shown above.

$$a_0 = \frac{q_f t_i - q_i t_f}{t_i - t_f} \qquad a_1 = -\frac{q_f - q_i}{t_i - t_f} \tag{8.18}$$

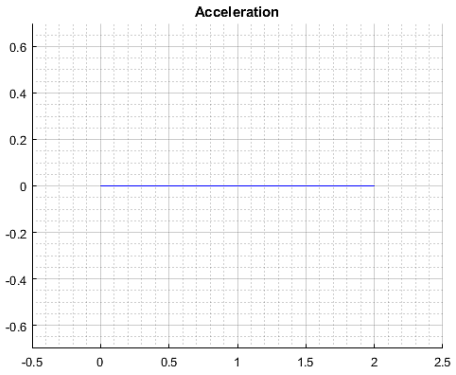
Using the above equations the following paths were generated in MATLAB.



(a) Position Plot



(b) Velocity Plot



(c) Acceleration Plot

Figure 8.3: Linear Trajectory Plot

Cubic Polynomial

The equation below represents the cubic trajectory generation (a polynomial of degree three).

$$q(t) = a_0 + a_1 \times t + a_2 \times t^2 + a_3 \times t^3 \tag{8.19}$$

$$\dot{q}(t) = a_1 + a_2 \times t + a_3 \times t^2 \tag{8.20}$$

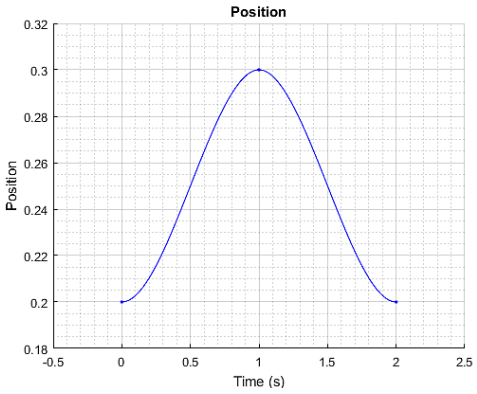
$$\ddot{q}(t) = a_2 + a_3 \times t \tag{8.21}$$

The variables a_0 to a_3 are solved using the same method used for the linear trajectory.

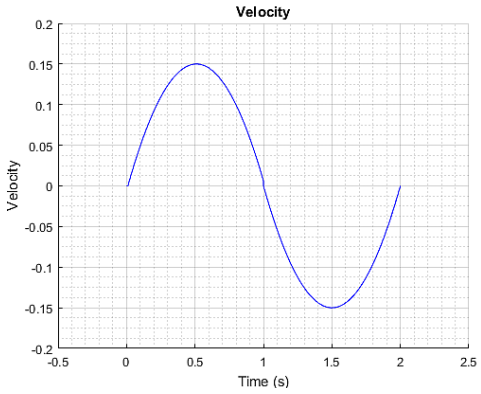
$$a_0 = \frac{q_f t_i^3 - 3q_f t_i^2 t_f + 3q_i t_i t_f^2 - q_i t_f^3}{(t_0 - t_f)(t_i^2 - 2t_i t_f + t_f^2)} \qquad a_1 = \frac{6(q_f t_i t_f - q_i t_i t_f)}{(t_0 - t_f)(t_i^2 - 2t_i t_f + t_f^2)} \tag{8.22}$$

$$a_2 = -\frac{3(q_f t_i - q_i t_i + q_f t_f - q_i t_f)}{(t_0 - t_f)(t_i^2 - 2t_i t_f + t_f^2)} \qquad a_3 = \frac{2(q_f - q_i)}{(t_0 - t_f)(t_i^2 - 2t_i t_f + t_f^2)} \tag{8.23}$$

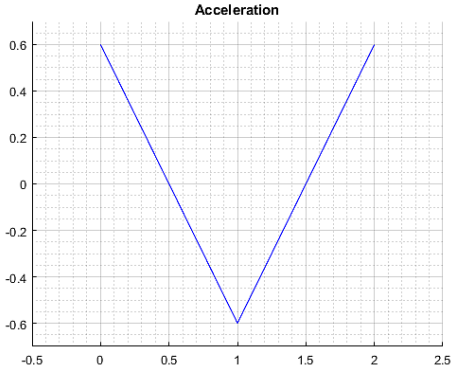
Using the above equations the following paths were generated in MATLAB.



(a) Position Plot



(b) Velocity Plot



(c) Acceleration Plot

Figure 8.4: Cubic Trajectory Plot

Quintic Polynomial

The equation below represents the quintic trajectory generation (a polynomial of degree five).

$$q(t) = a_0 + a_1 \times t + a_2 \times t^2 + a_3 \times t^3 + a_4 \times t^4 + a_5 \times t^5 \tag{8.24}$$

$$\dot{q}(t) = a_1 + a_2 \times t + a_3 \times t^2 + a_4 \times t^3 + a_5 \times t^4 \tag{8.25}$$

$$\ddot{q}(t) = a_2 + a_3 \times t + a_4 \times t^2 + a_5 \times t^3 \tag{8.26}$$

The variables a_0 to a_5 are solved using the same method used for the linear trajectory.

$$a_0 = \frac{q_f t_i^5 - 5q_f t_i^4 t_f + 10q_f t_i^3 t_f^2 - 10q_i t_i^2 t_f^3 + 5q_i t_i t_f^4 - q_i t_f^5}{(t_0 - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.27}$$

$$a_1 = -\frac{30(q_f t_i^2 t_f^2 - q_i t_i^2 t_f^2)}{(t_0 - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.28}$$

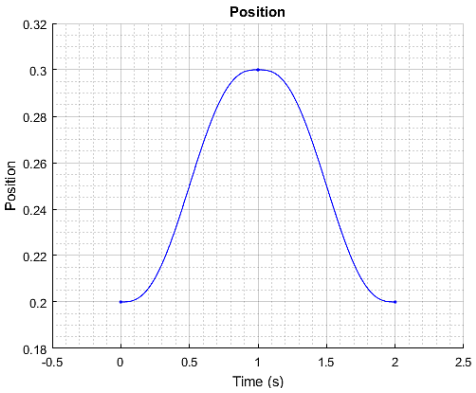
$$a_2 = \frac{30(q_f t_i t_f^2 + q_f t_i^2 t_f - q_i t_i t_f^2 - q_i t_i^2 t_f)}{(t_0 - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.29}$$

$$a_3 = -\frac{10(q_f t_i^2 - q_i t_i^2 + q_f t_f^2 - q_i t_f^2 + 4q_f t_i t_f - 4q_i t_i t_f)}{(t_i - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.30}$$

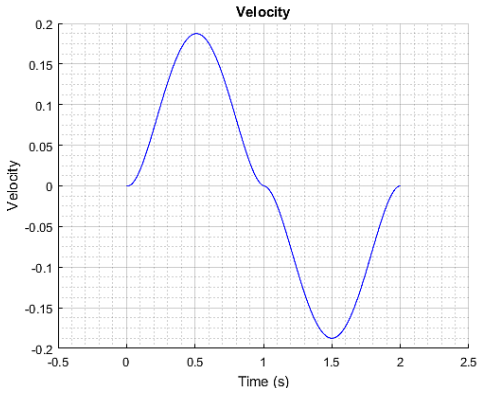
$$a_4 = \frac{15(t_i + t_f)(q_f - q_i)}{(t_i - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.31}$$

$$a_5 = -\frac{6(q_f - q_i)}{(t_i - t_f)^2(t_i^3 - 3t_i^2 t_f + 3t_i t_f^2 - t_f^3)} \tag{8.32}$$

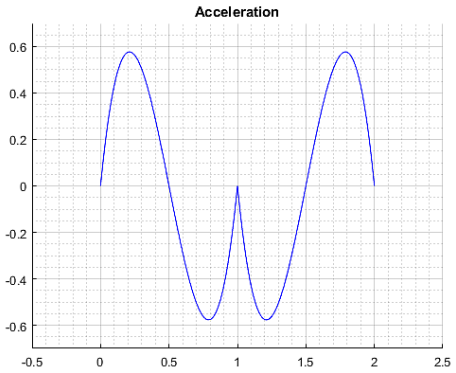
Using the above equations the following paths were generated in MATLAB.



(a) Position Plot



(b) Velocity Plot



(c) Acceleration Plot

Figure 8.5: Quintic Trajectory Plot

8.2.2 Single Sixth Order Polynomial with Via Point

In each of the above polynomial trajectories, two separate polynomials are required to generate the complete path. The computation is simpler if there is only one equation that takes the via point into account and generates the full path at once.

According to [48] a single sixth order polynomial is the optimum trajectory as it eliminates spikes in jerk at the start and end points which occur in lower order polynomials.

The equations for this single sixth order polynomial are shown below [48].

$$q(t) = a_0 + a_1 \times t + a_2 \times t^2 + a_3 \times t^3 + a_4 \times t^4 + a_5 \times t^5 + a_6 \times t^6 \tag{8.33}$$

$$\dot{q}(t) = a_1 + 2a_2 \times t + 3a_3 \times t^2 + 4a_4 \times t^3 + 5a_5 \times t^4 + 6a_6 \times t^5 \tag{8.34}$$

$$\ddot{q}(t) = 2a_2 + 6a_3 \times t + 12a_4 \times t^2 + 20a_5 \times t^3 + 30a_6 \times t^4 \tag{8.35}$$

The coefficients a_0 to a_6 are found using the initial, mid and final point constraints.

$$a_0 = q_i \qquad a_1 = 0 \qquad a_2 = 0 \tag{8.36}$$

For the general case of the hexapod, the mid point at the top of the stride is set to occur at $t_1 = \frac{t_f}{2}$ while t_i is zero. This led to simplified solutions for a_3 to a_6.

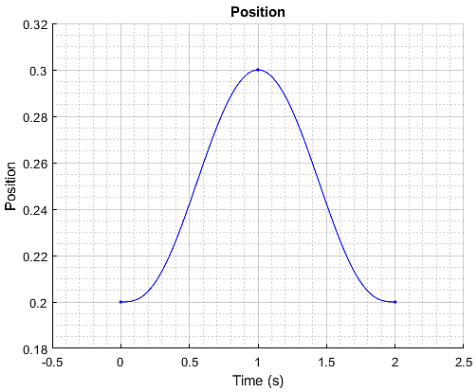
$$a_3 = \frac{2}{t_f^3}[32(q_1 - q_i) - 11(q_f - q_i)] \tag{8.37}$$

$$a_4 = -\frac{3}{t_f^4}[64(q_1 - q_i) - 27(q_f - q_i)] \tag{8.38}$$

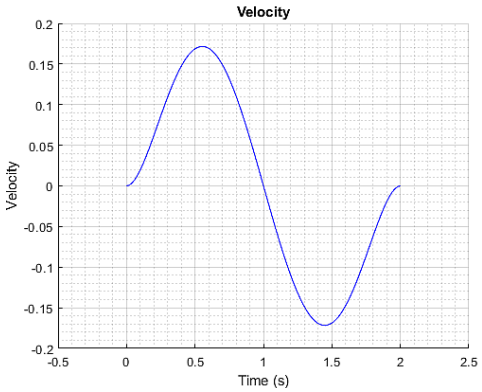
$$a_5 = \frac{3}{t_f^5}[64(q_1 - q_i) - 30(q_f - q_i)] \tag{8.39}$$

$$a_6 = -\frac{32}{t_f^6}[2(q_1 - q_i) - (q_f - q_i)] \tag{8.40}$$

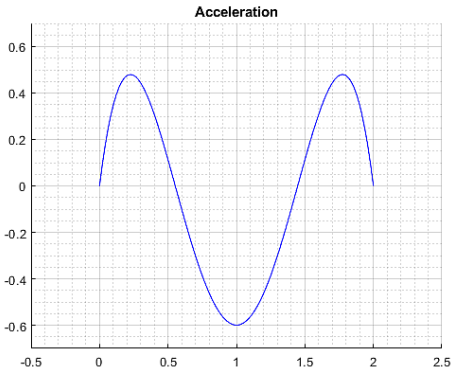
Using the above equations the following paths were generated in MATLAB.



(a) Position Plot



(b) Velocity Plot



(c) Acceleration Plot

Figure 8.6: Sixth Order Single Trajectory Plot

8.3 Discussion of Motion Planning

The motion planning algorithm was split into two sections. The foot position planning and the trajectory generation.

The foot position planning successfully generated starting, mid and end points for the trajectory generation algorithm to use. These points were generated using the given parameters of the radial distance from the center of the hexapod to the center of the stride line, the stride length, the step height, the crab angle and the pitch and roll of the hexapod.

Various different trajectory generation algorithms were explored, namely the cubic polynomial, the quintic polynomial and a single sixth order polynomial. The cubic and quintic polynomials require two separate equations to generate the full path of motion through the stride height via point while the single sixth order polynomial includes this via point in one equation. It is also clearly visible from the plots of the trajectory of each polynomial that the single sixth order polynomial is the smoothest and does not contain the acceleration spike or the 'flattening' of the velocity curve around the zero position that the quintic polynomial has in figure 8.5. This is mostly due to having one continuous equation as opposed to two separate equations that have been joined.

For this reason the single sixth order polynomial was selected as the trajectory generation algorithm for the leg of the hexapod. It was successfully implemented on the hexapod's micro-controller and was capable of propelling the hexapod in any given direction (based on the crab angle supplied to the algorithm).

The path generation algorithms were tested by walking the hexapod using these algorithms and observing the motion. This is shown during the testing section of this report.

9 Remote

In order for the hexapod to be useful in the real world it must be able to be remotely controlled. Relevant data including motion control instructions such as 'walk forward' and telemetry data must be communicated back and forth between the controlling device (i.e. a computer) and the hexapod robot. The wireless communication was achieved using a radio frequency serial module. This module transmits and receives serial data. Using a defined communication standard for sending and receiving messages all data can be transmitted to and received from the hexapod robot.

There are a number of different RF modules with varying communication frequencies. Common frequencies include very high frequency (VHF) of $136 - 174$Mhz which is best suited for outdoor applications where there are little to no obstructions between the transmitter and receiver and ultra high frequency (UHF) $403 - 470$MHz which is best suited for environments with obstructions [49].

For a robot that has search and rescue applications in mind, UHF is desired so that the communication will work through obstructions. A common UHF radio module for mobile applications is one that operates at 433MHz. These modules are low cost and small, making them ideal for the hexapod.

The module selected for use in this project is the RF1100-232 transceiver. This single module allows for both the transmission and reception of data using a serial (UART) interface.

It allows for point-to-multipoint transmission [50] which is beneficial as it allows the computer to transmit information to the hexapod and to any other devices fitted with another remote module, for example, the hexapod charging base discussed later. A number of these modules were readily available in RARL at UCT.

The module can be seen in figure 9.1 below.

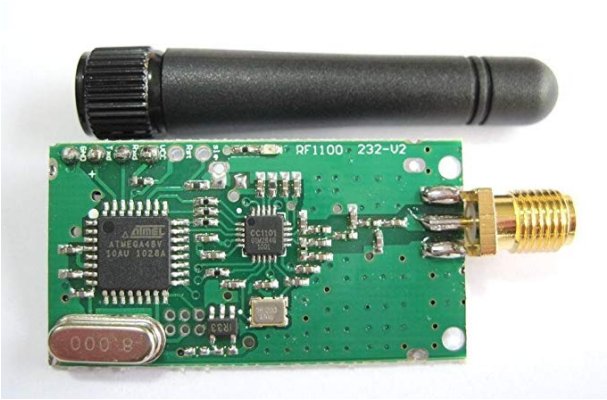


Figure 9.1: RF1101 Wireless Module [51]

Although these modules are ideal for remote communication they are 5V modules. To communicate with the micro-controller on the hexapod the 5V was dropped down to 3.3V using a voltage divider. After connecting the module through the voltage divider to the micro-controller and connecting a second module to a computer through a USB-to-serial converter, communication between the devices was achieved.

Unfortunately, data on this module was limited and as such range tests were performed to determine the range of the module.

The remote is configured with various different settings at start-up to ensure that it is communicating at the correct baud rate and on the correct channel. The commands to be sent to the module to set each setting of the module can be seen below [50]. Note that the portion of the command in brackets is what is changed to select different settings.

Read Module Configuration

0xA6 0x6A

where the module responds with its configuration as below.

0xA6 CHANNEL 0x64 BAUD POWER ID[1] ID[2]

Set Baud Rate

0xA3 0x3A (0x01/0x02/0x03)

where 0x01 is 4800 baud, 0x02 is 9600 baud and 0x03 is 19200 baud.

Set Channel Number

0xA7 0x7A (0x00-0xFF)

where (0x00-0xFF) is the desired channel number of 0 to 255.

Set Module ID

0xA9 0x9A (0x00,0xFF)

where (0x00,0xFF) is the desired module ID of 0 to 255 sent over 2 bytes.

Set Module Transmit Power

0xAB 0xBA (0x00/0x05/0x07/0x0A)

where (0x00/0x05/0x07/0x0A) is the module power in dbm (e.g. 0x0A is 10dbm).

A piece of software available at [50], simplified the initial configuration of the module by providing a GUI to change the module's settings. However the raw commands are to be used if the micro-controller needs to change any remote module settings during operation.

The configuration settings for the remote modules can be seen below.

Computer

Baud Rate: 19200

Channel: 1

ID: 1

Power: 10 dbm

Hexapod

Baud Rate: 19200

Channel: 1

ID: 2

Power: 10 dbm

Charging Base

Baud Rate: 19200

Channel: 1

ID: 3

Power: 10 dbm

9.1 Remote Test

The remote test entailed testing the range of the remote system and the response to obstacles between the hexapod and the host computer. This was done by setting the host computer to send a command to the hexapod every one second while moving the hexapod to multiple different locations and known distances from the host computer. The hexapod was monitored to see whether or not it received the data from the host computer at each location.

Table 9.1: Hexapod Remote Range Test

Distance (m)	Condition	Connection
20	Line of Sight (Outdoors)	True
60	Line of Sight (Outdoors)	True
100	Line of Sight (Outdoors)	True
110	Line of Sight (Outdoors)	True
120	Line of Sight (Outdoors)	False
20	Line of Sight (Indoors)	True
40	Line of Sight (Indoors)	True
60	Line of Sight (Indoors)	True
70	Line of Sight (Indoors)	False
20	Through Building Walls	True
40	Through Building Walls	True
50	Through Building Walls	True
60	Through Building Walls	False
10	Through Parked Cars	True
20	Through Parked Cars	False

9.2 Discussion of Remote

The remote was successfully implemented and two-way communication was achieved between the micro-controller and a computer. The voltage divider circuit can be seen in figure 11.7 in a later section.

The results above show that outdoors with line of sight the range of the remote is limited to ± 110m and line of sight indoors the range is limited to ± 60m. When the range was tested through building walls and floors by moving up floors in the building and moving into different rooms in the building the range was limited to ± 50m.

It is noted that large metal obstructions (such as cars) greatly reduce the range and in the case of multiple cars between the host computer and the hexapod the range was limited to ± 10m.

It was found that the remote module would, on occasion, lose its memory of its configuration. For this reason a function was developed on the hexapod micro-controller which checks the configuration of the remote module at start-up and if it is not correct it would go through the process of setting each parameter. It then reads the configuration again to confirm that everything has been set correctly before continuing.

10 IMU Stabilisation

Inertial measurement units, or IMUs, are sensors which comprise of accelerometers, gyroscopes, magnetometers and microprocessors used to process and collect measurements of the linear and angular acceleration of the system to which it is connected.

10.1 Accelerometer

An accelerometer is an independent instrument which measures the acceleration of an object in $m.s^{-2}$ or G-force. They are low power devices and therefore can be powered by a micro-controller. The measurement range of each accelerometer varies but common values include $\pm 2G, \pm 4G, \pm 8G$ and $\pm 16G$, however they can have a range as high as $\pm 250G$ [52].

There are two main types of accelerometers: capacitive and piezoelectric.

Capacitive accelerometers output a voltage depending on the distance between internal, charged plates. As the device is accelerated the distance between these plates varies and from this the acceleration is determined [53].

Piezoelectric accelerometers contain a crystal which produces a voltage when tension or compression is applied to it. A mass attached to a spring compresses the crystal when it is accelerated and this produces a voltage from which the acceleration is determined [53].

Accelerometers are cost effective and easy to implement into a system, however they are prone to noise if not carefully selected and processed.

10.2 Gyroscope

A gyroscope determines angular velocity using the Coriolis effect. Its output is generally in rad/s^2 and this can be integrated to determine angle of rotation of the gyroscope [54]. Three axis gyroscopes allow for the determination of the angle in each axis, namely pitch[6], roll[7] and yaw[8]. Gyroscopes are prone to drift and as such are normally fused with an accelerometer to achieve consistent data readings over time. The yaw cannot be determined using a gyroscope alone and sensor fusion with a magnetometer is also required.

10.3 Magnetometer

A U-shaped beam made from a ferrous metal with two piezoresistive sensors placed on each end indicate differential strain depending on its orientation with the Earth's magnetic field [55]. This produces a varying voltage with varying angular position. Three axis magnetometers allow for angular position sensing in all directions.

[6] Rotation around the x-axis
[7] Rotation around the y-axis
[8] Rotation around the z-axis

10.4 The Xsens IMU

An Xsens Motion Tracker MTi-28A33G35 was used as the IMU unit for the hexapod due to its availability in the Robotics and Agents Research Lab and its precision in determining drift free, three dimensional orientation as well as calibrated three dimensional linear acceleration, angular acceleration and magnetic field data [56]. It was selected over conventional IMU modules such as the MPU-9250 [57] due to its robust and dust proof case as well as its simple setup using its own software GUI, MT Manager [58].

It has an embedded processor capable of performing necessary sensor fusion and calculating roll, pitch and yaw in real time using a Kalman Filter. The Xsens Kalman Filter algorithm is known as XKF-3 and uses data from the gyroscopes, accelerometers and magnetometers to compute an optimal three dimensional orientation estimate with a high degree of accuracy and no drift for static and dynamic movements [56].

Figure 10.1: XSens IMU

10.5 Stabilisation

Stabilisation, and pitch and roll control, is important to allow for sensors, such as a camera, to be mounted onto the hexapod's body. The pitch and roll of the hexapod can be set to a particular angle and even while walking on a slope, that angle is maintained. This is beneficial as it allows for the pointing of a camera in a fixed direction without the need for more motors to move the camera independently.

Note that stabilisation refers to the rejection of a base motion to achieve a stable system. For pitch and roll this would normally be done using the angular rate as opposed to the angular position as this produces a control system with a higher bandwidth. The stabilisation implemented on the hexapod does not take into account the angular rate, simply the angular position of the hexapod's body.

As such, here the word stabilisation is used to describe the control of the pitch and roll of the hexapod and the ability to set it to a fixed value (relative to a flat ground) which will not change if the ground under the hexapod changes. This means that the hexapod does in fact reject a base motion to produce a stable pitch and roll, however it does so by controlling the pitch and roll angles as opposed to the angular rate.

10.5.1 Feedback Control

Feedback control is a control mechanism that uses readings of the output of a control loop to adjust the input to the loop in order to achieve a desired output. A basic, closed loop feedback control loop can be seen in figure 10.2 below.

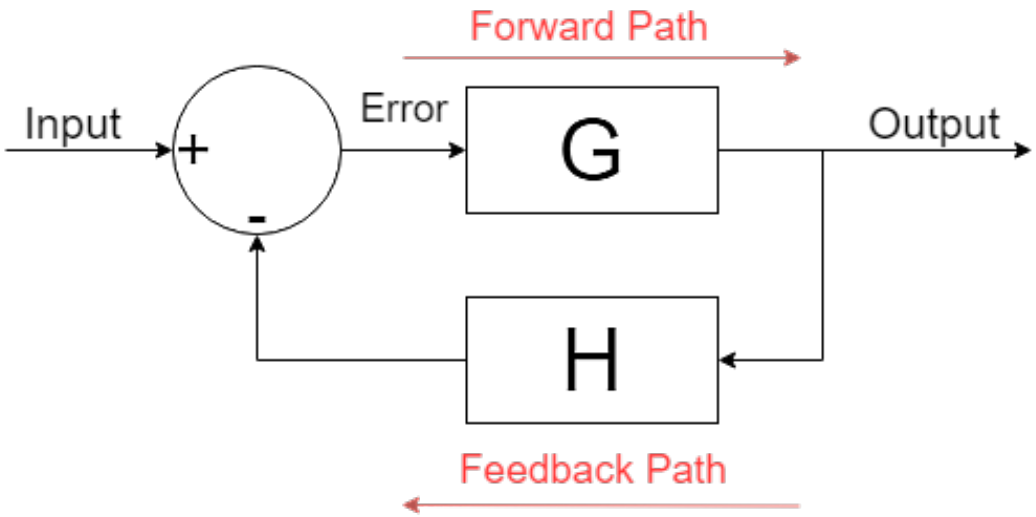


Figure 10.2: Basic Negative Feedback Loop

A feedback loop can be either positive feedback or negative feedback. Positive feedback is used to increase the size of the error term by summing the input and output values while negative feedback is used to decrease the error term by subtracting the input and output values. A negative feedback loop is said to be stable because it sends the error term to a constant value [59].

10.5.2 PID Control

A proportional-integral-derivative (PID) controller is a commonly used negative feedback controller. It is comprised of a proportional controller, and integral controller and a derivative controller.

Proportional Controller

The proportional controller depends on the current error in the system. Therefore if the proportional gain is increased, the error term decreases at a faster rate. The proportional gain is denoted as K_P. The proportional term is given by $P = K_P \times e$ where e is the error term in the loop.

A high proportional gain leads to a large change in the output for a given change in the input and if the gain is too high this can lead to the system becoming unstable [59]. If the proportional gain is small then the output has a small response to a change of the input. If the gain is too low the system may not respond to a change in input. Increasing this gain reduces the rise time of the loop and aids in reducing the steady state error.

Integral Controller

The integral controller depends on both the size of the error and the duration of the error. It is the sum of the instantaneous error over time. The integral controller aids in eliminating the stead state error, however it makes the transient response worse [59]. The integral term is given by $I = K_I \int_0^t e(t)dt$ where, as in the proportional controller, $e(t)$ is the error term with respect to time and K_I is the integral gain.

Derivative Controller

The derivative controller depends on the rate of change of the error. It is used to slow the rate of change of the output of the control loop which in turn increases the stability of the system and reduces the overshoot. This improves the transient response. The derivative term is given by $D = K_D \frac{de(t)}{dt}$ where K_D is the derivative gain.

A helpful table describing the effects of increasing each of the above controller gains has on a system is shown below [59].

Table 10.1: PID Controller Gain Effects

Parameter	Rise Time	Overshoot	Settling Time	Steady State Error
$\mathbf{K_P}$	Decrease	Increase	Small Increase	Decrease
$\mathbf{K_I}$	Small Decrease	Increase	Increase	Large Decrease
$\mathbf{K_D}$	Small Decrease	Decrease	Decrease	Minor Change

A common structure of the PID controller is the parallel connection of each of the above controllers as shown in figure 10.3 below [59].

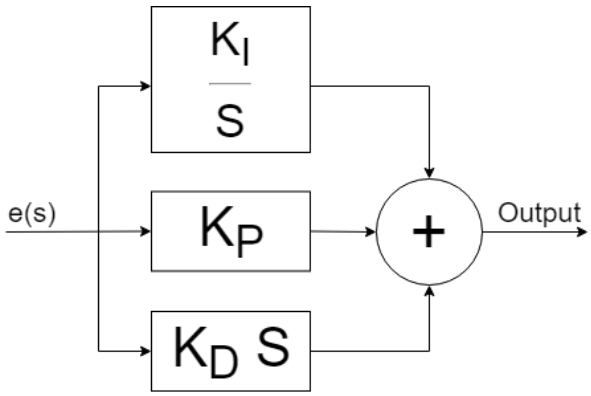


Figure 10.3: Parallel PID Controller

In order to implement a PID controller on the hexapod robot for pitch and roll control a digital PID controller is developed. For pitch control the digital PID controller takes in the difference between the desired pitch (set point) and the actual pitch and outputs a pitch value to which the hexapod must go to therefore decreasing the error until it is zero.

The proportional controller can be represented mathematically as simply the product of the error term and the K_P gain.

The integral controller is defined as the cumulative sum of the previous error terms multiplied by the integral gain K_I.

The derivative controller is defined as the difference between the current error and the previous error, multiplied by the derivative gain K_D.

10.5.3 Implementing the PID Controller

The PID controller was developed in C++ so that it can be implemented onto the hexapod's micro-controller. It uses a recursive algorithm to determine the proportional, integral and derivative components of the controller. The equations used in the algorithm are shown below [60].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de}{dt} \tag{10.1}$$

The error term $e(t)$ is defined as:

$$e(t) = r - y \tag{10.2}$$

where r is the controller set point and y is the parameter being controlled (in this case the current pitch or roll).

This is shown in the form of a control loop block diagram below.

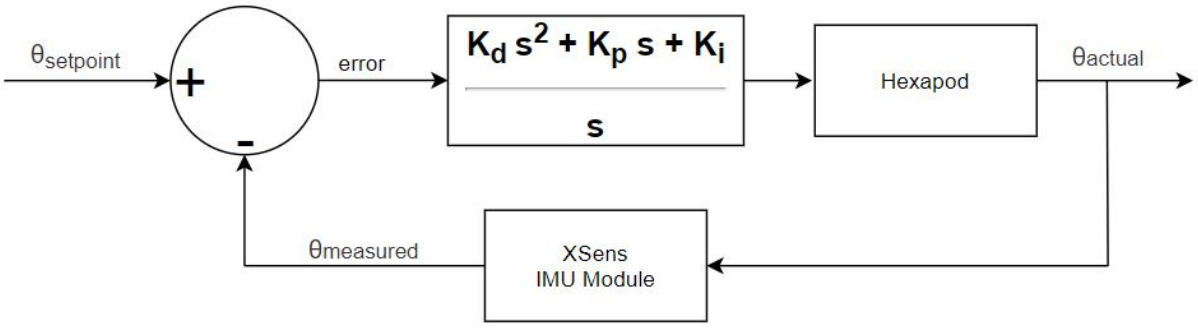


Figure 10.4: Final PID Control Loop Block Diagram

The above block diagram represents the control loop for one attitude angle (e.g. pitch). Two of these control loops are run in parallel to control both pitch and roll.

In the C++ algorithm this is implemented as follows:

1. Initialise an integral term precursor i and a previous error term e_p to zero.
2. Initialise K_p, K_i, K_d and dt to their respective values. Note that dt is the time between PID calculations.
3. Calculate the current error at time t, $e = r - y$.
4. Calculate the proportional term $P = K_p e(t)$.
5. Calculate the cumulative integral term precursor $i = i + edt$.
6. Calculate the integral term $I = K_i i$.
7. Calculate the derivative term $D = K_d \frac{(e-e_p)}{dt}$.
8. Calculate the total PID output $u = P + I + D$.
9. Update the previous error term $e_p = e$.
10. Repeat steps 3 to 9 on each time interval dt.

This process is executed every 100ms in a timer interrupt and the pitch and roll of the hexapod is adjusted to the total PID output, u, after each iteration.

Using the desired hexapod pitch and roll output from the above PID controller, a height offset is applied to each leg of the hexapod in order to set the body attitude. This is done using the equation below.

$$\Delta z = x \times tan(pitch) + y \times tan(roll) \tag{10.3}$$

where x, y and z are the coordinates of a particular foot of the hexapod.

The above equation for Δz supplies each foot with a change in z required to set the Hexapod's pitch and roll. There are many possible ways for the pitch and roll to be adjusted by moving all 6 legs, however this equation assumes that each leg must adjust the pitch and roll equally, therefore distributing the load of each leg equally.

10.5.4 Tuning the PID Controller

Two PID controllers were developed for the hexapod. One for pitch control and one for roll control. To start, the roll controller was switched off while the pitch controller was tuned. The tuning was done using the Ziegler-Nichols method [61].

To do this the integral and derivative gains are set to zero. The proportional gain is then increased from zero until the system begins to oscillate at a constant amplitude. The gain at this point K_u as well as the oscillation period, T_u, are then used to set the PID gains for system according to the table below [59].

Note that the Ziegler-Nichols method is designed to be used on linear systems. In the real world the hexapod is a nonlinear system [41] and as such the Ziegler-Nichols method is used only as a starting point for tuning. The PID gains were then adjusted from this point until the stability of the system was optimized.

Table 10.2: Ziegler-Nichols Method

Control Type	$\mathbf{K_P}$	$\mathbf{K_I}$	$\mathbf{K_D}$
Normal PID	$0.6K_u$	$\frac{2K_P}{T_u}$	$\frac{K_P T_u}{8}$
Some Overshoot	$0.33K_u$	$\frac{2K_P}{T_u}$	$\frac{K_P T_u}{3}$
No Overshoot	$0.2K_u$	$\frac{2K_P}{T_u}$	$\frac{K_P T_u}{3}$

The above process was performed on the hexapod and the graph of pitch and roll over time was recorded. The graph below shows the pitch and roll of the hexapod over time as the K_p value is increased to the point of oscillation of the hexapod.

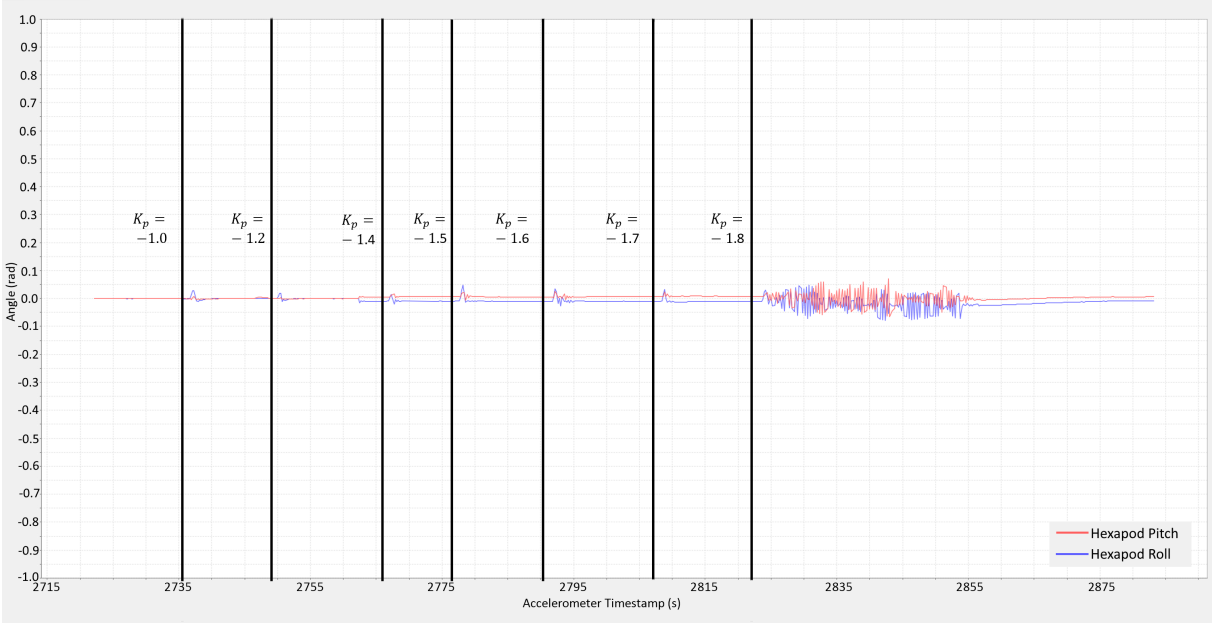


Figure 10.5: Hexapod PID Tuning

The graph shows that at a critical K_p value of 1.8 the hexapod begins to oscillate. The oscillations were focused on to determine the oscillation frequency.

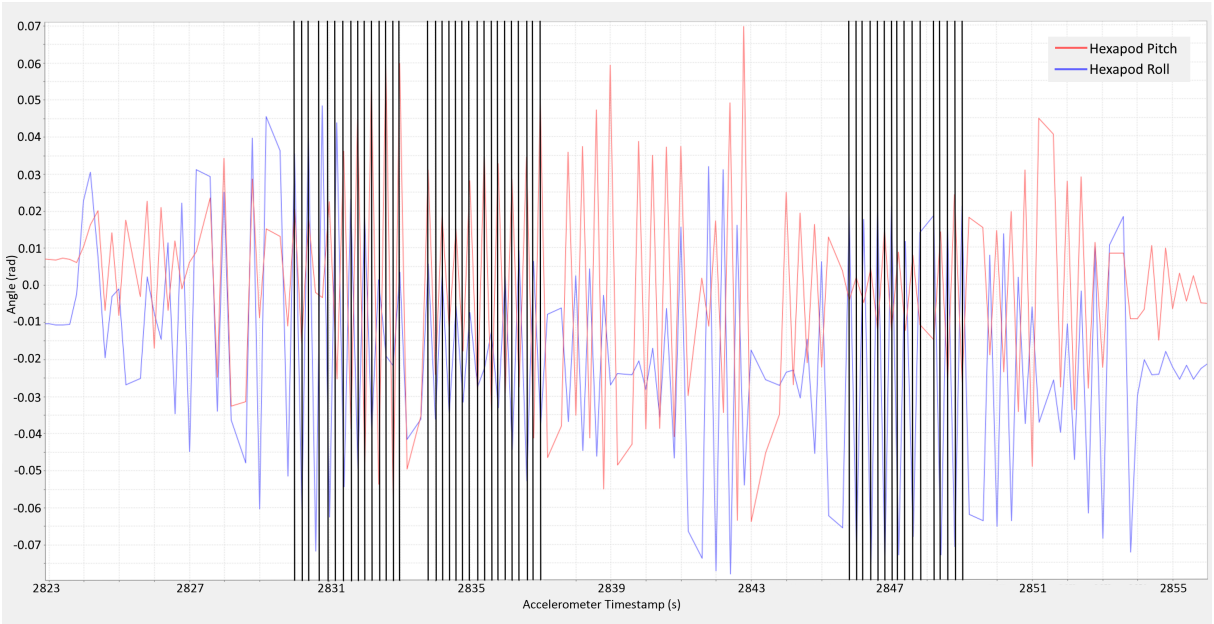


Figure 10.6: Hexapod PID Critical K_p

In graph 10.6 lines were drawn at the peak amplitude of each oscillation to determine the period. The period was measured in three different places during the oscillation with the average of these taken. The final measured period of oscillation was 0.2 seconds.

Using the critical K_p value, K_u, and the period of oscillation T_u in conjunction with table 10.2 the values for K_p, K_i and K_d were calculated using the normal PID configuration.

$$K_p = 0.6 \times 1.8 = 1.08 \tag{10.4}$$

$$K_i = \frac{2 \times 1.08}{0.2} = 10.8 \tag{10.5}$$

$$K_d = \frac{1.08 \times 0.2}{8} = 0.027 \tag{10.6}$$

Using the above PID gain values the hexapod's stabilisation was tested and the resulting graph of pitch and roll is shown below.

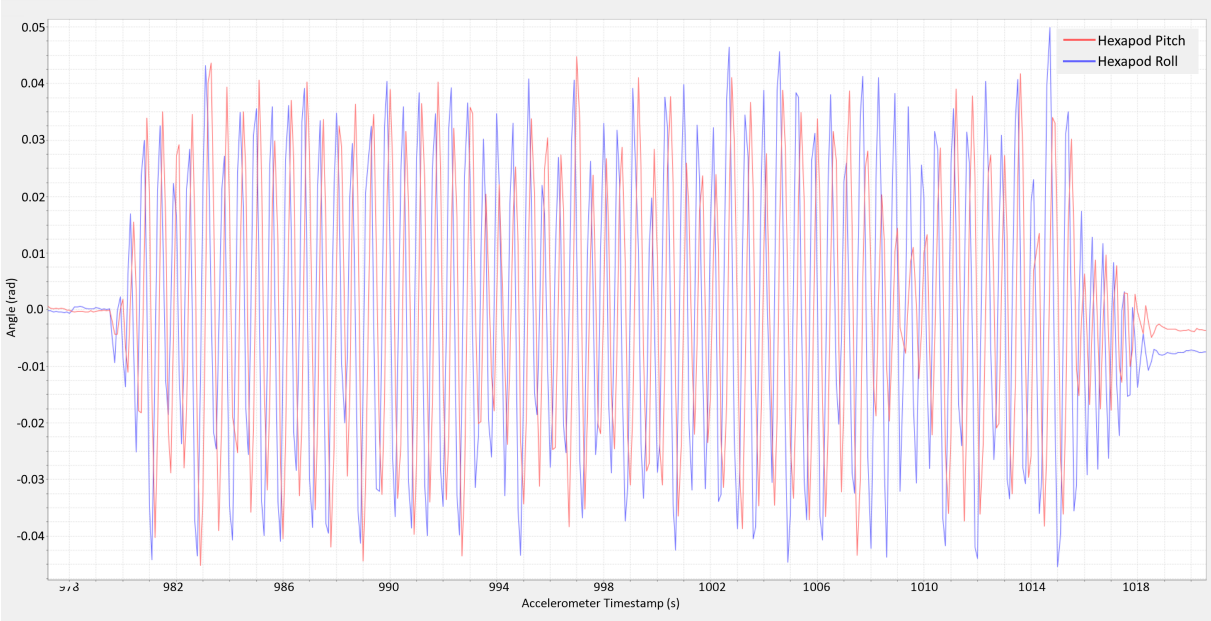


Figure 10.7: Hexapod PID Initial Values

It can be seen in the graph above that the control loop with the gain values calculated using the Ziegler-Nichols method is not stable. The oscillations begin with no disturbance to the hexapod and they never settle. The hexapod moves back and forth continuously in both pitch and roll. Note that the points at the beginning and end of the oscillatory portion of the graph is before the PID controller is enabled and after it is disabled.

The PID gains were then adjusted manually, using the Ziegler-Nichols values as a starting point, until the hexapod became stable. The gains were chosen so as to minimize the settling time, the overshoot and the oscillations. According to table 10.1 in order to decrease the settling time and overshoot the integral gain must be reduced.

By reducing the integral gain incrementally and testing the response by angling the table that the hexapod was standing on by a known angle of ± 0.3 rad a final, acceptable stabilisation was found. A plot of the hexapod's pitch and roll over time with these final PID gains can be seen in figure 10.8 below.

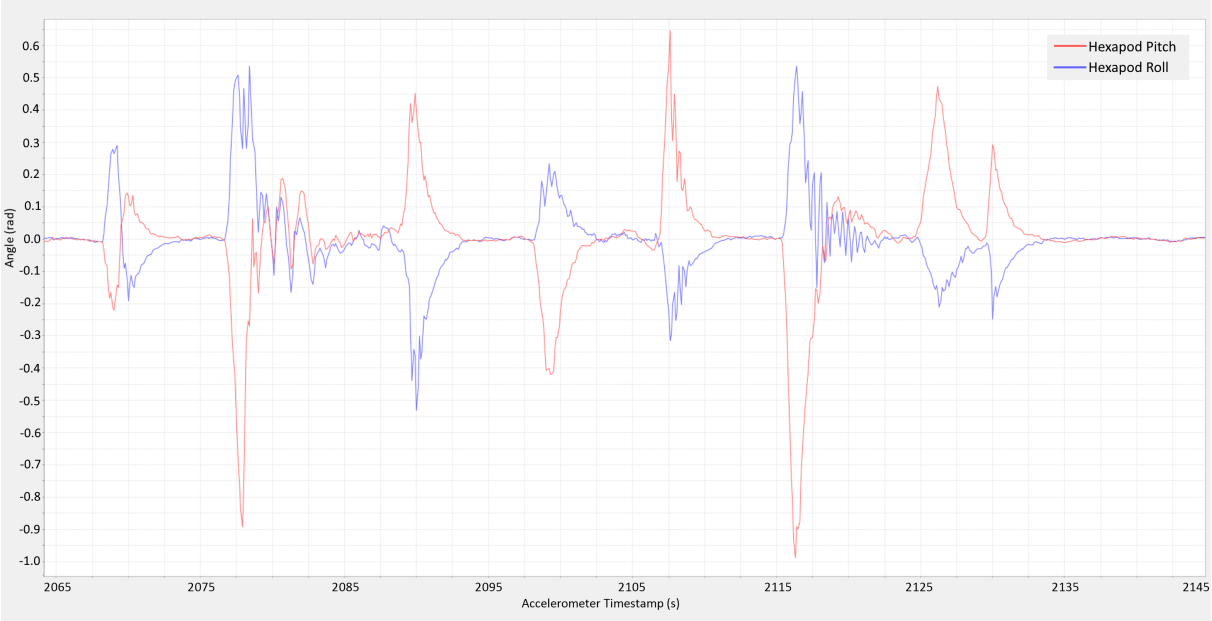


Figure 10.8: Final PID Gain Test

Final values for the gains are shown below.

$$K_p = 1.10 \tag{10.7}$$

$$K_i = 2.21 \tag{10.8}$$

$$K_d = 0.032 \tag{10.9}$$

The same test was performed with the IMU stabilisation switched off and with the controller switched on with a K_p of 1 and the other gains set to zero to show the effect of the properly tuned controller.

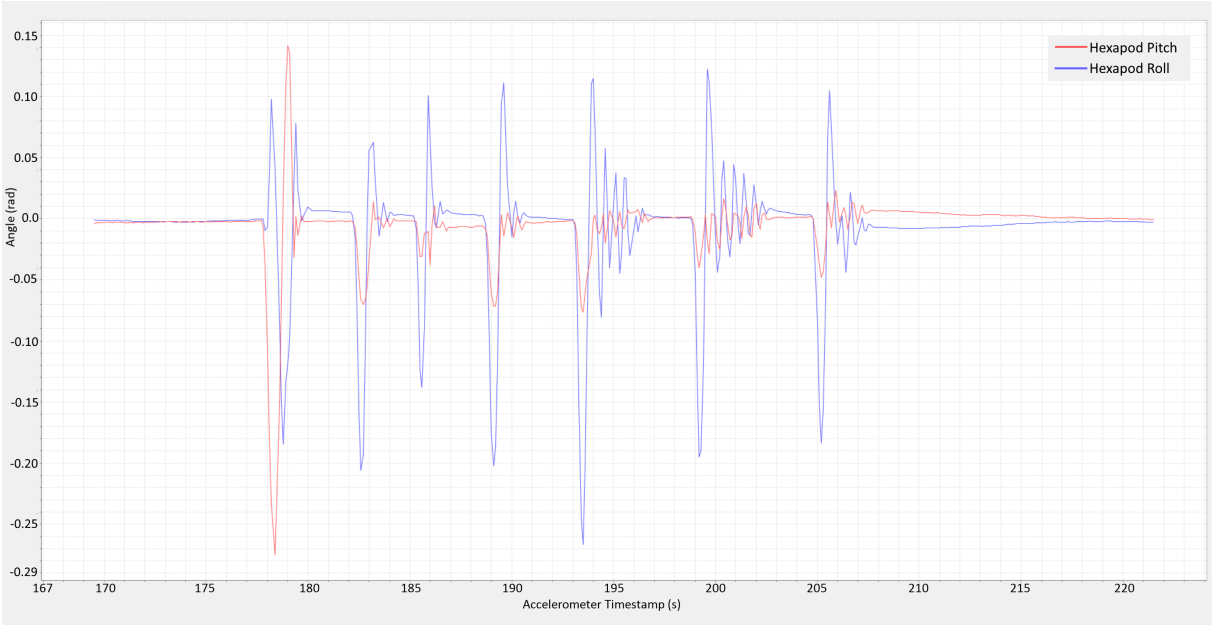


Figure 10.9: PID with $K_p = 1$, $K_i = K_d = 0$

Figure 10.9 above shows that without the PID controller and simply a P controller with a gain of 1 the hexapods pitch and roll oscillates before settling and has a steady state error of ± 0.01 rad. Comparing this to figure 10.8 which is properly tuned for stabilisation shows that the properly tuned PID controller is effective in reducing, if not eliminating, both steady state error, overshoot and oscillations of the hexapod's pitch and roll.

Figure 10.10 below shows that without any stabilisation the pitch and roll of the hexapod simply matches that of the slope angle it is standing on, with a steady state error.

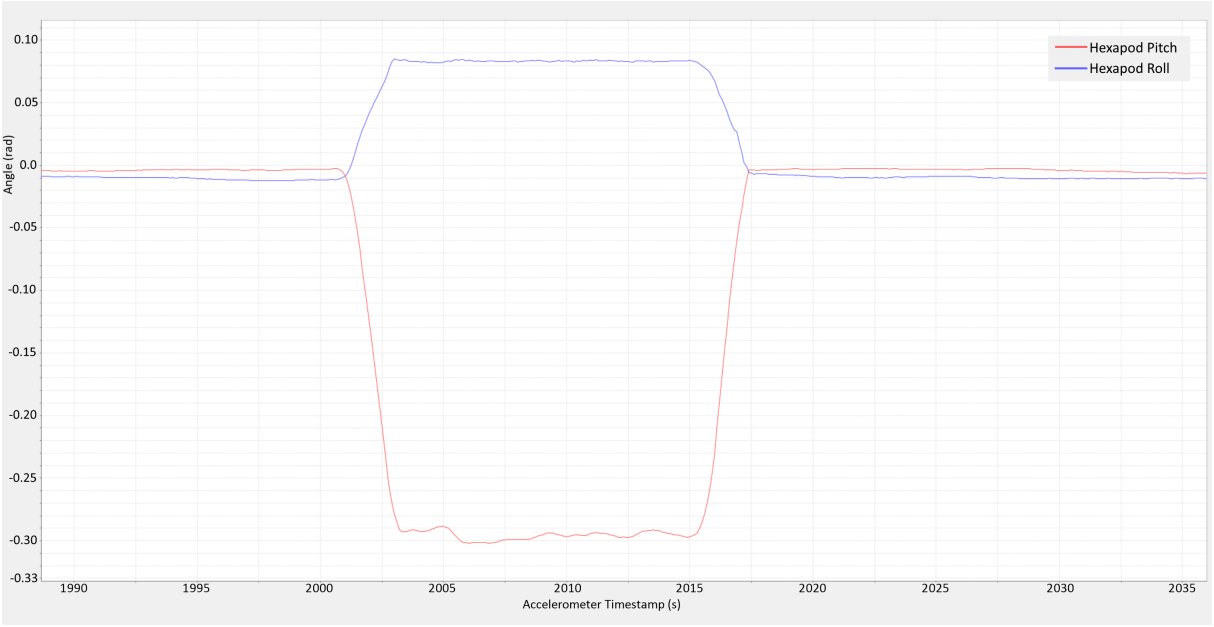


Figure 10.10: Stabilisation Switched Off Test

In the case when the IMU stabilisation is switched on and correctly tuned as in figure 10.8 it is clear that the pitch and roll settles back to zero over time and the steady state error is eliminated.

10.6 Discussion of IMU

The IMU module was successfully implemented with the hexapod's micro-controller. Communication between the devices was achieved and accurate pitch and roll data was read from the IMU by the micro-controller.

This allowed for the development of a PID controller for pitch and roll which after proper tuning allows for the setting of pitch and roll values and the stabilisation of the body of the hexapod to said values.

The final PID gain values are shown below.

$$K_p = 1.10 \tag{10.10}$$

$$K_i = 2.21 \tag{10.11}$$

$$K_d = 0.032 \tag{10.12}$$

These values were successfully tuned to achieve minimum overshoot, oscillations and steady state error.

Graphs were plotted of the hexapod with the properly tuned controller, an improperly tuned controller, a simple P controller and with no controller at all. These graphs were used to verify the functionality of the properly tuned PID controller.

11 Circuit Design

The main controller for the hexapod was originally selected to be the STM32F051C6 micro-controller, however after extensive use it was found that this micro-controller was not powerful enough to control all elements of the hexapod. It did not have enough flash memory to store the C++ program required. For this reason the micro-controller was upgraded to the larger STM32F407VGT6.

In order to interface with the motors, IMU and remote additional circuitry was required. Specifically a circuit to convert from RS485 communication to UART was needed for the motors, conversion circuitry from RS232 to UART was required for the IMU and a voltage level adjuster was required to interface with the remote.

The circuits were designed and developed first on a breadboard and later on veroboard for a more permanent solution. Finally, a printed circuit board was designed and manufactured for the final circuit for the hexapod.

11.1 STM32F407VGT6 Core Circuit

The circuit diagram for the core micro-controller components is shown in figure 11.1 below.

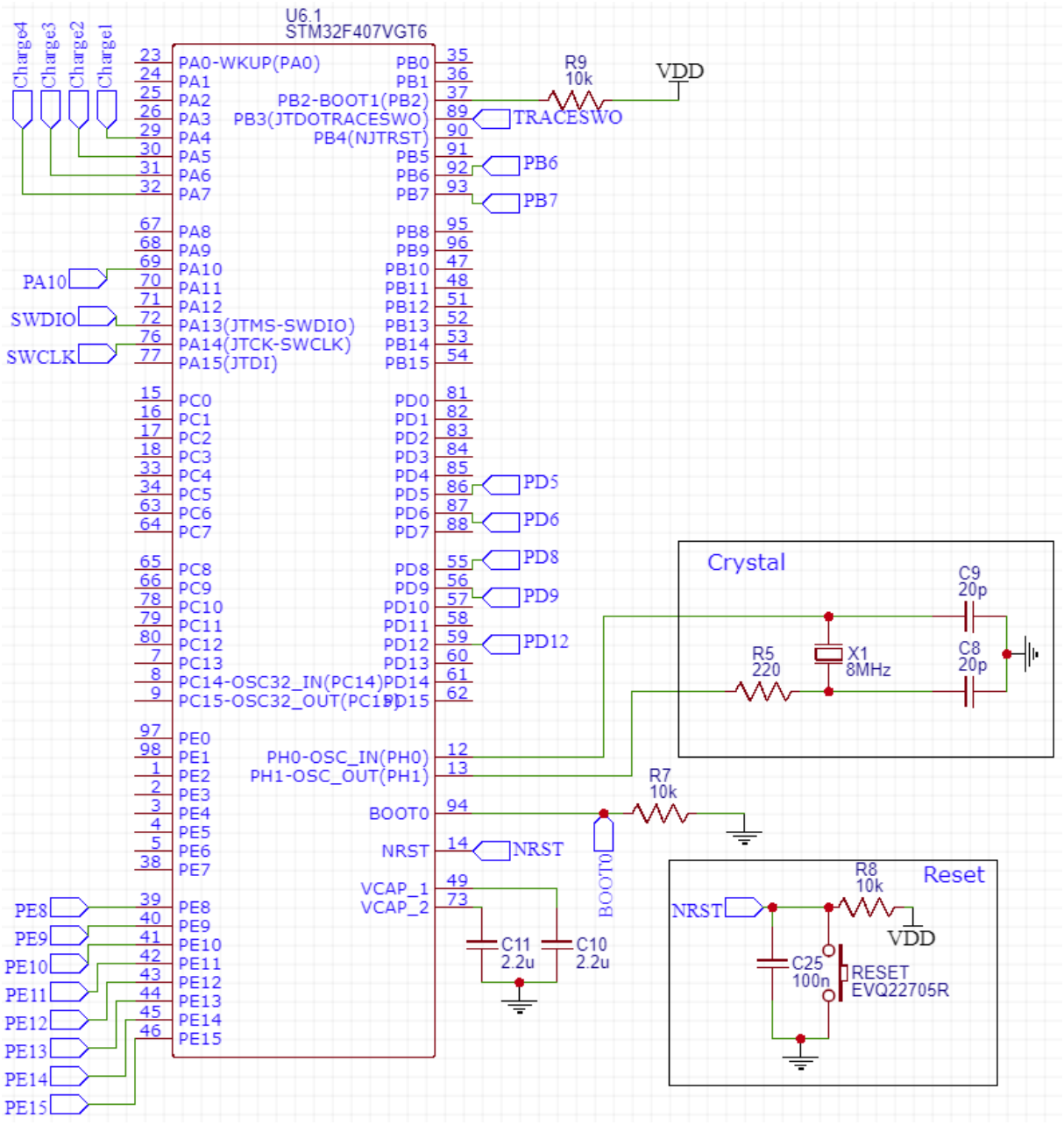


Figure 11.1: STM32F407VGT6 Core Circuit Diagram

The micro-controller requires a voltage of 3.3V to operate and as such a voltage regulator is needed to produce this. The circuit diagram for the 3.3V regulator is shown in figure 11.2 below. Note that VDD is the 3.3V output of the regulator.

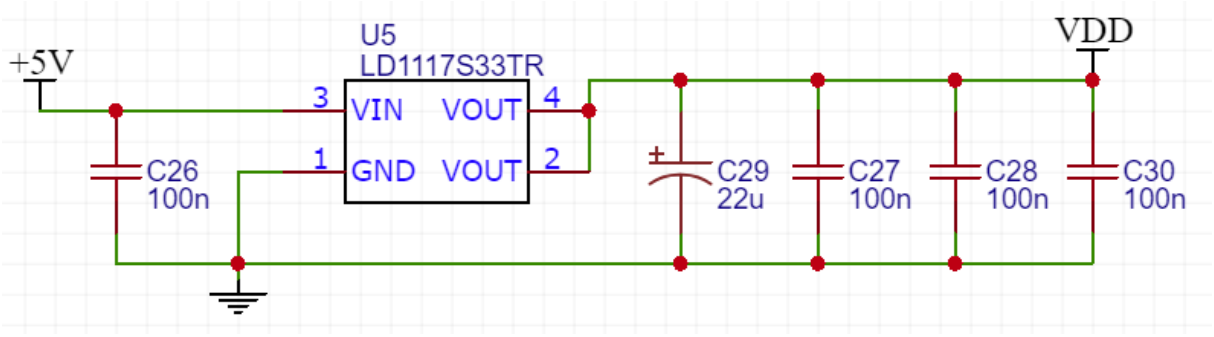


Figure 11.2: 3.3V Regulator Circuit Diagram

The power component of the micro-controller is shown in the circuit diagram below.

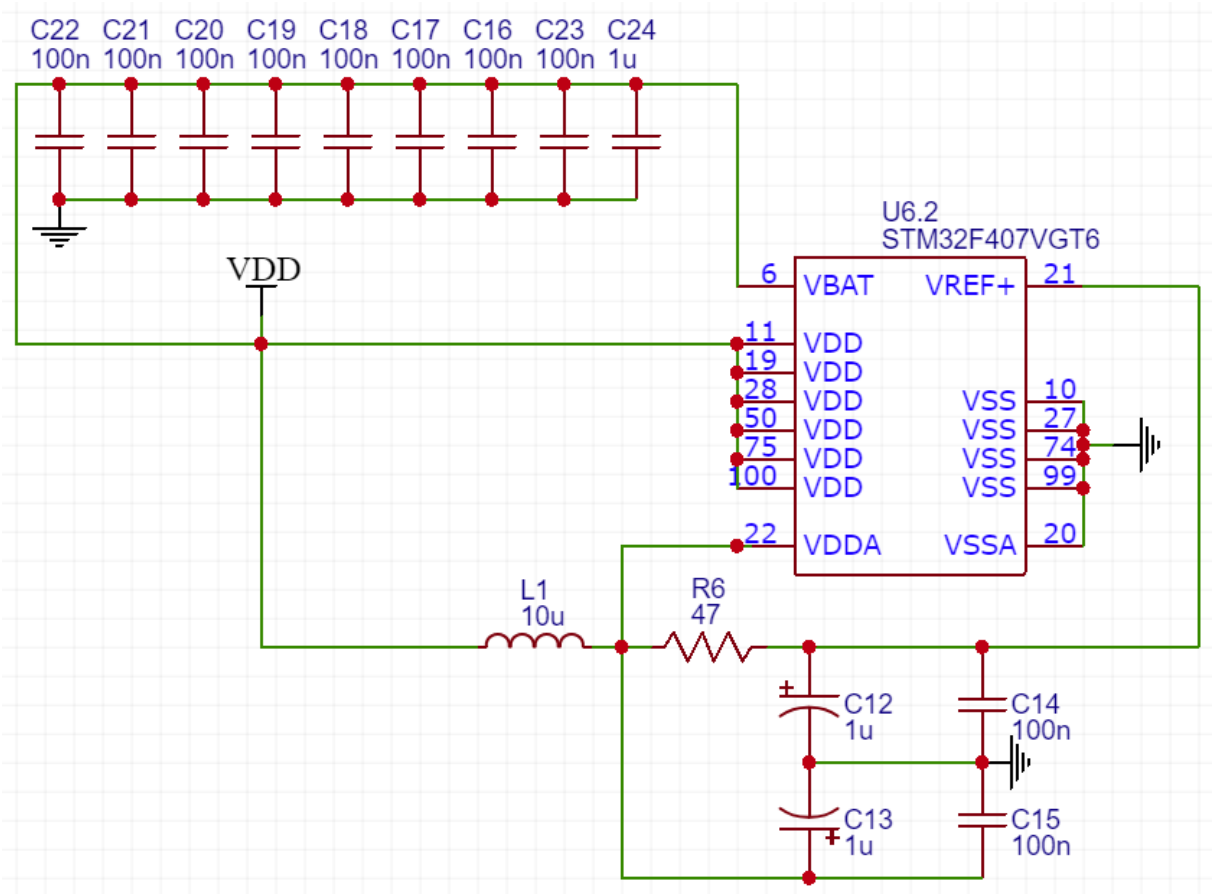


Figure 11.3: STM32F407VGT6 Power Circuit Diagram

11.2 Motor Communication Circuit

According to the Dynamixel Motors user manual [25] the recommended circuit diagram to communicate with a UART device is as shown in figure 11.4 below.

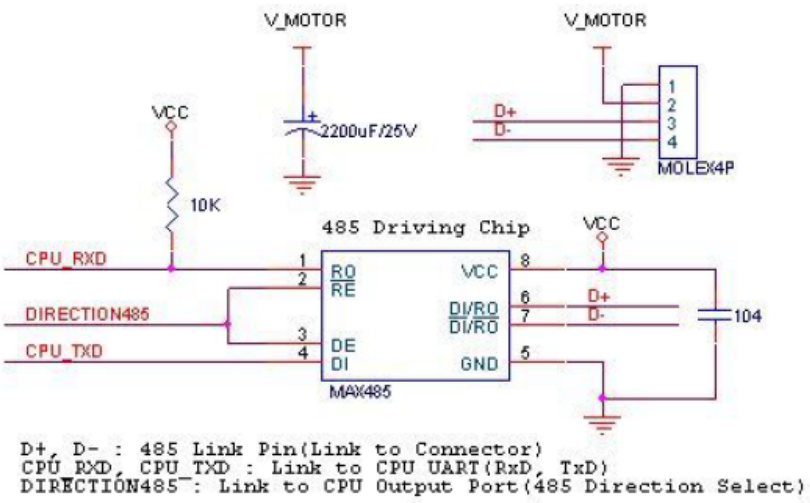


Figure 11.4: Recommended RS485 to UART Converter Circuit

This was adjusted slightly to use an integrated circuit that was more easily available. The final circuit diagram for the motor communication can be seen in figure 11.5 below.

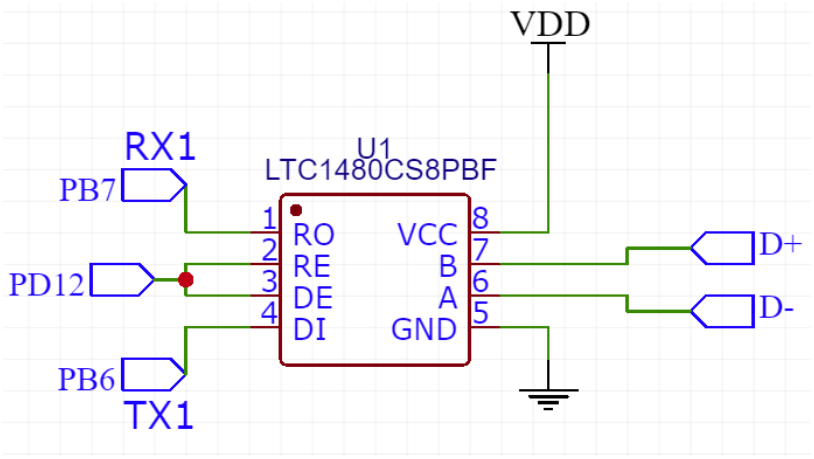


Figure 11.5: Final RS485 to UART Converter Circuit

In the above circuit diagram the nets PB6, PB7 and PD12 connect to the respective pins on the microcontroller and nets D+ and D- connect to the motors communication pins. Pins PB6 and PB7 are used for UART communications while pin PD12 is used to switch the LTC1480 chip between transmit-to-motors and receive-from-motors mode by pulling the pin high or low respectively. The motors are connected in a daisy chain fashion from one to another.

11.3 IMU Communication Circuit

The Xsens IMU communicates using RS232 serial protocol which operates from +3V to +15V and -3V to -15V logic levels [62]. The micro-controller must communicate using UART serial protocol which operates from 0V to 3.3V logic levels. A conversion circuit is required to allow for successful communication between these two devices. It is based on the MAX232 integrated circuit and can be seen in the diagram below.

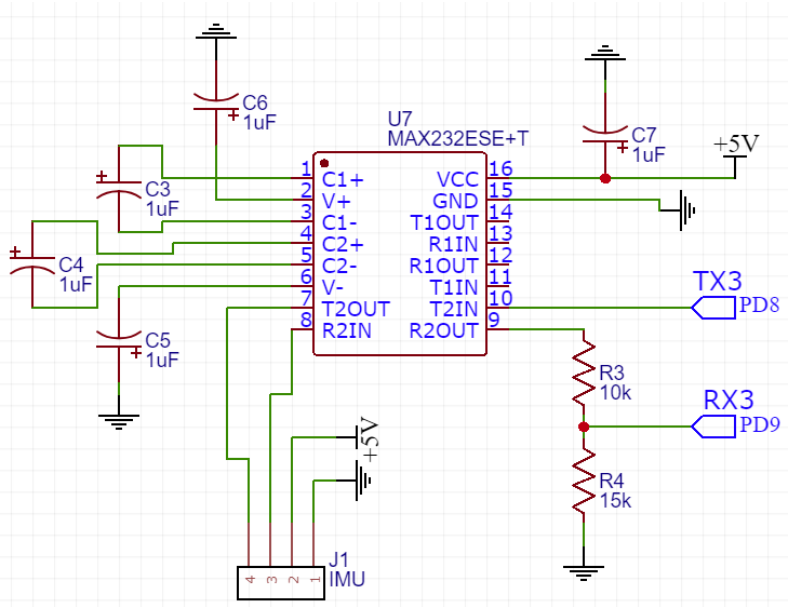


Figure 11.6: Final RS232 to UART Converter Circuit

In the circuit diagram above the pins PD8 and PD9 are connected to the UART of the micro-controller.

11.4 Remote Communication Circuit

The remote operates at 5V logic levels and therefore needs a simple voltage divider to lower the output of the remote to 3.3V. No circuitry is needed on the input to the remote as it recognises 3.3V as logic level high correctly.

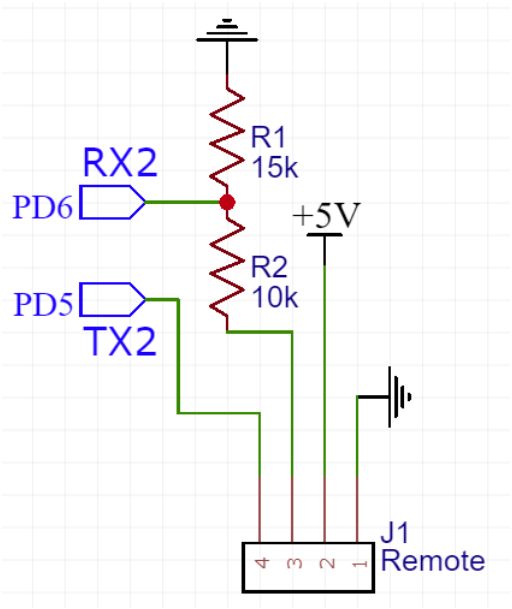


Figure 11.7: Remote Output Voltage Divider Circuit

11.5 Motor Switch Circuit

The hexapod has been designed in conjunction with wireless charging circuity. During the charging of the hexapod there cannot be a current draw from the batteries of more than 150mA or the charging circuit cannot determine the charge level of the batteries. For this reason a switch is needed to disconnect the motors from the battery during charging.

The switch needs to be capable of withstanding up to 10A of peak current. A relay was investigated however due to its large size it was dismissed as an option. In order to produce a circuit that is small enough to fit onto a PCB within the hexapod, MOSFETS were used. Two MOSFETS in parallel were used to achieve the required current capabilities. The circuit diagram can be seen in figure 11.8 below.

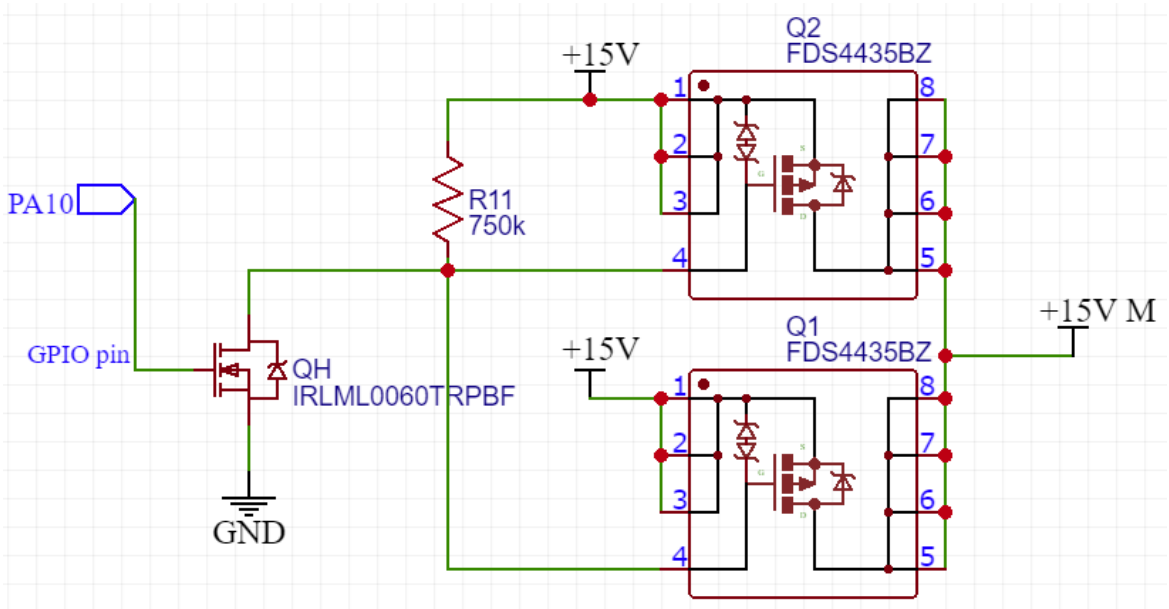


Figure 11.8: Motor Power Switch

In the above circuit the micro-controller controls the switch using GPIO pin PA10 as seen connected to the driver MOSFET QH.

This circuit was designed by Sven Weihe during his undergraduate thesis working on the hexapod in 2019 [7].

11.6 Additional Connections

In addition to the above circuits, connectors are included to provide a connection to the charging circuit, the SWD pins of the micro-controller for programming and GPIO pins to be used for future expansion as well as the main power input to the control circuit.

11.6.1 Power Input

A three pin power input connector is required to provide the main control board with 5V for power to the integrated circuits and voltage regulator and 15V for power to the motors. A phoenix connector was used as it has higher current carrying capabilities than a standard molex. This is needed for the 15V connection to the motors.

11.6.2 Charging Circuit Connection

A four pin connector was added to GPIO pins on the micro-controller. This is to be connected to the charge controller for the battery system. These four pins are used to read the state of charging and determine if charging is complete as well as read the current battery voltage.

11.6.3 Expansion Pins

An expansion header was added connecting to pins PE8 to PE15 on the micro-controller to allow for future GPIO expansion. Note that these pins do not have access to the ADC or other communication ports on the micro-controller.

11.6.4 SWD Programming

A six pin header was added connecting to the following pins on the micro-controller to allow for programming of the chip.

- SWDIO (PA13)
- SWCLK (PA14)
- TRACESWO (PB3)
- BOOT0 (Pin 94)
- NRST (Pin 14)
- GND

A circuit with the ST-Link debugger chip was developed to interface with the SWD pins and a computer. This circuit diagram can be seen in figure 11.9 below.

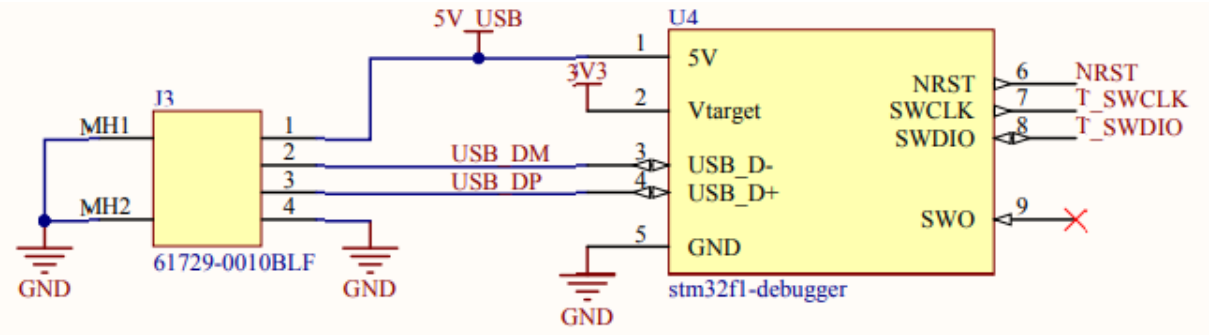
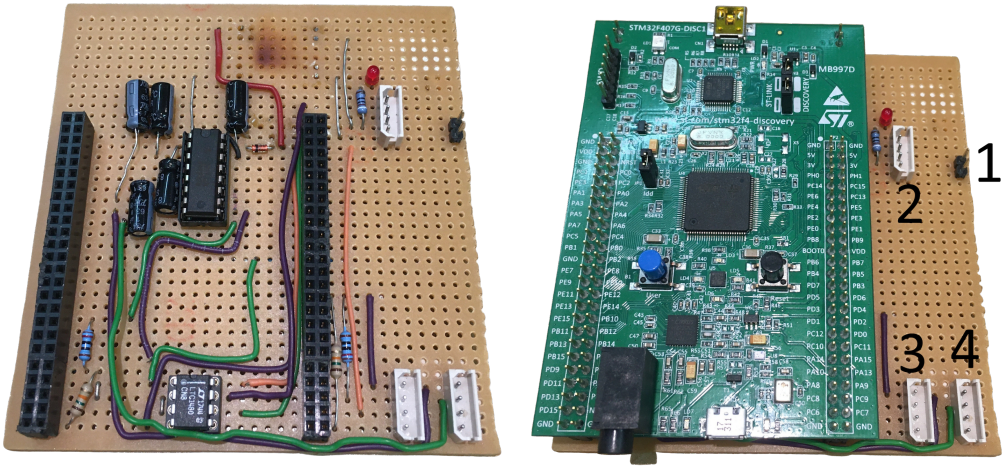


Figure 11.9: ST-Link Debugger Circuit [63]

11.7 Prototype Circuit Board

A prototype circuit board was developed using veroboard. This was done to be able to test each component of the circuit before designing the PCB. It was designed such that the STM32F4-Discovery board could be connected to it. The prototype circuit board can be seen in figure 11.10 below.



(a) Prototype Board without Discovery (b) Prototype Board with Discovery

Figure 11.10: Prototype Circuit Board

In the circuit board above, power is connected to connection 1, the motors are daisy chained onto connection 2, the remote is connected to connection 3 and the IMU is connected to connection 4.

11.8 Printed Circuit Board Design

A printed circuit board (PCB) with all of the above circuits and connectors was designed using EasyEDA [64]. The final, full circuit diagram can be seen in the figure below.

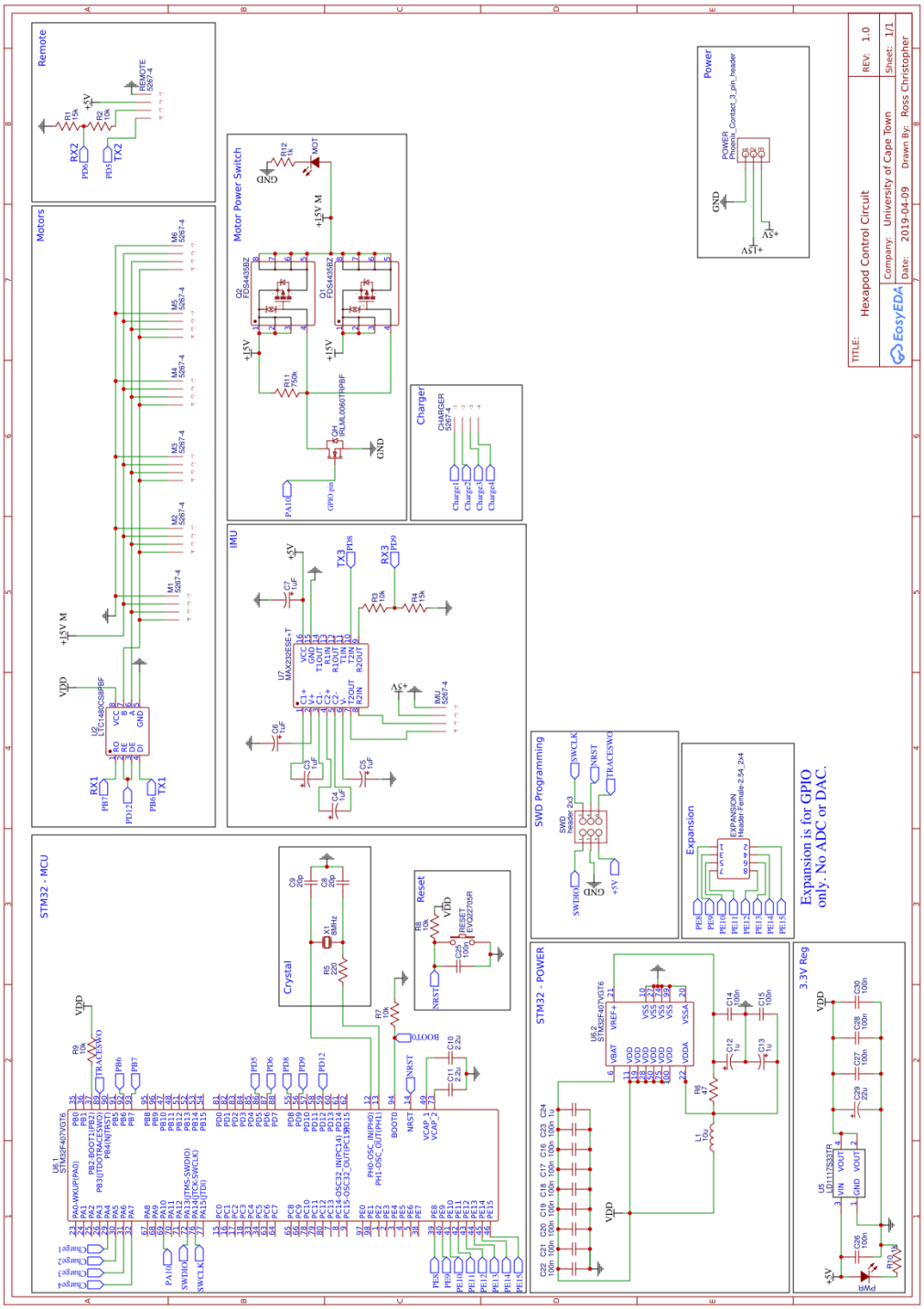
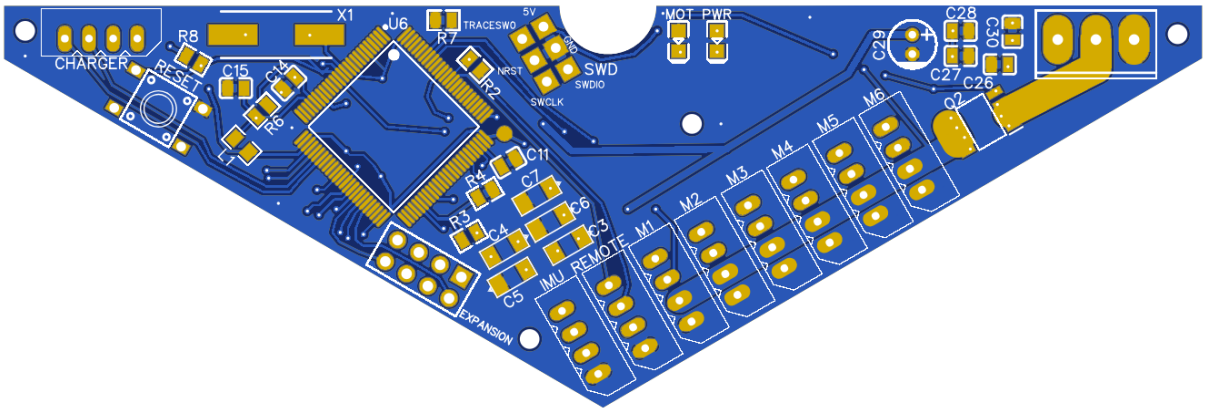
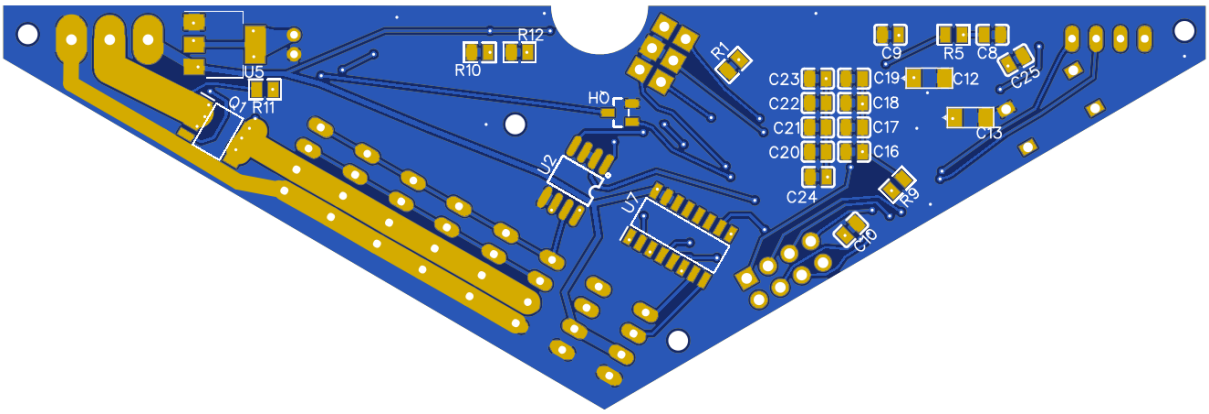


Figure 11.11: Final Complete Circuit Diagram

A rendering of the PCB can be seen in figure 11.12 below.



(a) Top Layer



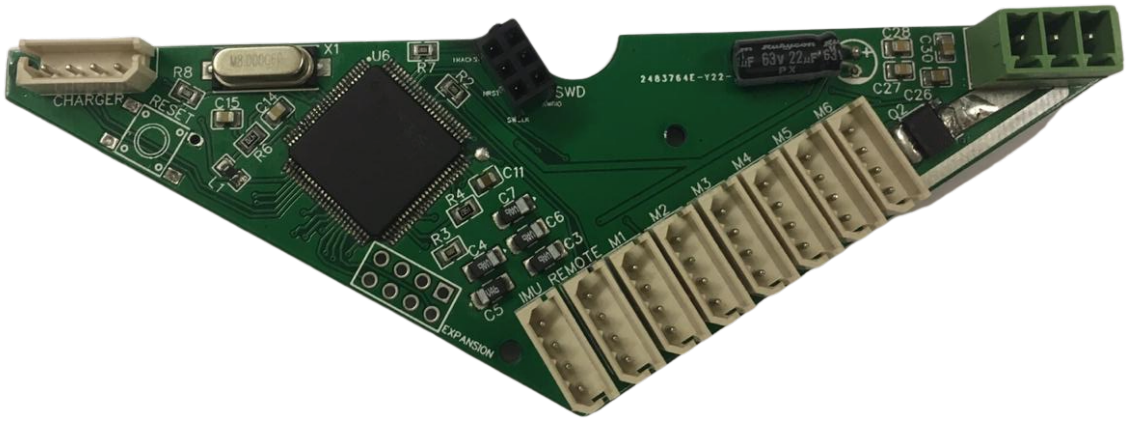
(b) Bottom Layer

Figure 11.12: PCB Design

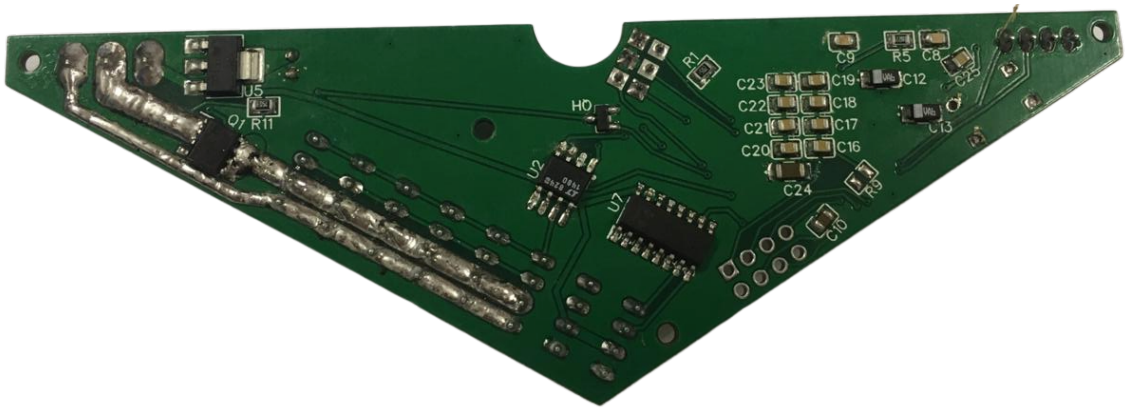
The above PCB was designed using components with the SMD-0805 package so that they can be easily hand soldered in place. Traces that are required to carry higher currents have been exposed so that solder can be deposited onto them to increase the current carrying capabilities.

11.9 Completed Circuit Board

The surface mount components were soldered onto the PCB using a solder reflow oven in RARL at UCT. Larger components were soldered by hand. The final completed board can be seen in the figure below.



(a) Top Layer



(b) Bottom Layer

Figure 11.13: Completed PCB

Extra solder was added to the power tracks for the motors as seen on the left of the bottom layer above. This was done to increase the current carrying capabilities of the tracks.

11.10 Discussion of Electrical Design

The main hexapod control PCB was designed based on a successful prototype board. Various features were added to it that did not exist on the prototype board in order to add functionality and improve performance.

The STM32 micro-controller can be successfully programmed using the SWD pins supplied on the PCB and a ST-Link debugger, although programming would have been made easier by the inclusion of a 3.3V power pin in the SWD header to allow for powering up the micro-controller from the debugger. Without this pin the board must be powered by a 5V supply through the main power connector, if the batteries are not connected.

The board can successfully communicate with the motors and control them. The motor power switch which is used to switch power to the motors on or off using a GPIO pin on the micro-controller worked correctly in switching the motor power, however when the motor power was switched on, the UART 2 port being used for the remote control failed. If the motors were switched on by placing an external 3.3V onto the enable pin everything worked as expected. This was strange and the conclusion that was made was that noise from the motors was causing the micro-controller malfunction. The GPIO pin signal was viewed on the oscilloscope and no noise was obviously visible however the problem was evident.

A possible solution to this problem was adding a signal diode in series with the GPIO pin of the micro-controller to prevent any signals from disturbing the pin. However this was not successful.

It was later determined that this motor switch was not needed as the originally designed 15V motor voltage regulator circuit did not function correctly [7] and a store bought replacement regulator was fitted. This store bought regulator had a built in enable/disable pin which when interfaced to one of the GPIO expansion pins on the PCB successfully enabled and disabled power to the motors. The motor switch chips were therefore removed, as they were no longer necessary, and the PCB was adjusted by bridging the motor power across where the switches used to be with wire.

With this problem solved, the board functioned correctly and was able to communicate with the motors, the remote and the IMU at the same time. It successfully read the battery status and voltage and transmitted this information to the user through the remote which was received by the control GUI and displayed.

The board communicated successfully with the charging base, sending battery voltage and charge status information as well as commands to start and stop charging to which the charging base responded correctly [7].

One notable error in the design of the hexapod PCB was the reset button. The polarity of the reset button was not correct and as such when the button was fitted the micro-controller's reset pin was constantly pulled to ground, placing it in an infinite reset loop. This was resolved by simply removing the reset button from the PCB, however a redesign of the board to correct this would be helpful as it is useful to be able to reset the micro-controller by pressing this button instead of having to power cycle the whole hexapod. This problem did not cause any issues with the functionality of the hexapod, however it was an inconvenience.

12 Programming the Hexapod and GUI

The hexapod robot was programmed using object orientated C++. This was done so that each component of hexapod is a separate class which allowed for modular code design with simple implementation of functions. It also helped to make the code easier to read, understand and keep track of all functions.

The STM32F4 HAL library [65] was used to program the micro-controller. This allowed for a high level of abstraction which aided in a faster development time.

12.1 GPIO Pins

Various GPIO pins are used to interface with peripheral devices such as UART and the ADC. These pins are outlined below.

12.1.1 GPIO A

Pin 4

PA4 is used as a GPIO output to enable to reading of the current battery voltage. A switch is placed on the charging PCB which disconnects the battery from the micro-controller ADC when it is not being used. This was done to limit the power dissipation from the battery. By setting PA4 to high the switch is turned on and reading of the battery voltage is possible.

Pin 5

PA5 is an analog input to the micro controller's ADC which is connected to a scaled battery voltage (using a voltage divider connected to the battery and the switch as mentioned above) from 0V to 3.3V. This value is used to determine the current battery voltage.

Pin 6 and 7

PA6 and PA7 are GPIO digital inputs which are used to determine the current status of battery charging. The table below describes the state of charging relative to the two pin values.

Table 12.1: Battery Charger Status Pins

Battery Status	PA6	PA7
Not Charging	HIGH	HIGH
Fault	HIGH	LOW
Charging Constant Current	LOW	HIGH
Charging Constant Voltage	LOW	LOW

Pin 10

PA10 is used to enable or disable the power to the servo motors. During charging the hexapod must draw no more than 150ma [66] for the charging circuit to function correctly. If all the motors are connected to power, even with no load on them, they draw over 1A. For this reason it must be possible to disconnect the motors from power during charging using a digital switch connected to PA10.

12.1.2 GPIO B

Pins PB6 and PB7 are used as the UART 1 transmit and receive pins. These pins connect to the motors and allow for communication between the controller and the motors.

12.1.3 GPIO D

Pins PD5 and PD6 are used as the UART 2 transmit and receive pins. These pins connect to the wireless transceiver and allow for communication between the hexapod, the charging base and the controlling computer.

Pins PD8 and PD9 are used as the UART 3 transmit and receive pins. These pins connect to the XSens IMU and allow for the reading of pitch and roll information from the IMU.

12.1.4 GPIO E

GPIO E has been left as expansion pins for future use. The pins PE8 to PE15 can be used to add future functionality to the hexapod without the need for developing a new PCB.

12.2 Implementing Position Control

The position control was implemented by using the inverse kinematics derived previously to calculate the respective motor angles in radians of each motor on the leg for a defined XYZ foot position. These values were set to the theta position of the Leg object for each leg.

This position in radians for each motor was then translated into the bits that were to be written to the motor's goal position register.

12.2.1 Motor Position Bits Calculation

The conversion of the motor position from $-150°$ to $150°$, to $0°$ to $300°$ and to then convert this angle into a 16 bit unsigned integer is shown in equation 12.1 below. Note that this equation is in radians.

$$\theta_{bits} = 195.569(\theta + 2.618) \tag{12.1}$$

In the above equation θ_{bits} is the position value in byte form and θ is the position value from -2.61799 to 2.61799 radians. The constant 195.569 comes from the fact that 1024 is the highest possible number in a 16 bit unsigned integer. If the maximum angle is 5.23598 radians (300 degrees) then $\frac{1024}{5.23598} = 195.569$.

This final position value is then rounded down to the nearest integer value and using the following bitwise operations, converted into an array of 2 bytes.

uint8_t byte1 = (uint16_t) θ_{bits} $\&$ 0xFF
uint8_t byte2 = (uint16_t) θ_{bits} $>>$ 8

where $\&$ is a bitwise and function and $>>$ is a bitwise right shift function.

12.3 Implementing Speed Control

In order to calculate the rotational speed of each motor given a global hexapod XYZ speed the differential kinematics of the legs was used.

This value is then converted into a 2 byte array using the method shown in the previous section.

12.4 Implementing Path Generation

The path generation algorithm is based off the maths described in section 8. First the path generation parameters are defined with default values when the hexapod starts up. This values are shown below.

$$\text{Radial Distance} = 250\text{mm} \tag{12.2}$$

$$\text{Direction} = -10 \tag{12.3}$$

$$\text{Stride Length} = 50\text{mm} \tag{12.4}$$

$$\text{Step Height} = 50\text{mm} \tag{12.5}$$

$$\text{Body Height} = 140\text{mm} \tag{12.6}$$

Using these parameters the starting, middle and end points of each leg are first calculated and then the path generation algorithm is run to generate an array of points for each leg.

12.5 Software Calibrating The Legs

The legs needed to be calibrated so that they were better aligned with one another. This was done by doing a 'tap test' with each link of the leg. First the horizontal hip motor was rotated until the torque on that motor spiked, indicating that it had collided with the hexapod body which is a fixed reference point. This position at which it collided was recorded. The vertical hip motor was then rotated until the first leg link came into contact with the top of this motor. This was another fixed reference point. This position was recorded. The final, third motor in the leg was then rotated until the third link collided with the second link, another fixed reference point. This position was recorded.

Each of the fixed reference points described above is in the same position for all legs on the hexapod. After completing this calibration of the first leg, the position values gathered for each motor were defined as the zero position. From this point the calibration procedure was repeated for each leg and the position values for all motors were recorded. The difference between each motors position value and the position value of the first leg's motors (the zero position) was calculated and this value was used as a software offset constant to apply in the position control code.

This calibration technique was repeated 10 times and the average offset value for each motor was taken. These offset values can be seen in table 12.2 below.

Table 12.2: Motor Calibration Offset Constants (rad)

Motor Number	Leg 1	Leg 2	Leg 3	Leg 4	Leg 5	Leg 6
Motor 1	0	0	0	0	0	0
Motor 2	0	0.12101	0.10056	0.10567	0.15680	0.02045
Motor 3	0	0.01534	0.00511	0.01363	0.0	0.02556

An alternative to the process above is mounting micro switches and using the press of the micro-switch as a trigger for the 'tap test'. This was considered however it was deemed to be less accurate than the method described above. Ensuring that each of the 18 micro-switches required for calibration are installed in the exact same place on each leg would have been difficult, however the position of links and motors on each leg is already known to be the same as the parts were all machined to match. Furthermore, additional programming would have been required on 18 GPIO pins to read the state of each micro-switch. This makes the micro-switch method more complicated, and potentially more inaccurate, than the torque spike method above.

12.6 Implementing IMU Communication

The XSens IMU was first connected to a computer using a USB to serial converter cable. Using the MT Manager software [58] the IMU was calibrated and setup to output calibrated pitch, roll, yaw and a timestamp only when it receive a data request command. Without this setup the IMU would constantly send data to its serial port. This would cause the hexapod's receive data interrupt to be called constantly and other processes (such as motor communication and remote control) would not have enough time to complete correctly.

When the hexapod is ready to receive orientation data from the IMU it sends the 'request data command' to the IMU. This is command is a 5 byte array and is structured as follows.

0xFA 0x01 0x34 0x00 0xCB

After receiving this command the IMU responds with its data in the following format.

Note that '[]' defines how many bytes are used by the parameter, if there is no '[]' it can be assumed that it is 1 byte only. This notation is kept throughout this report.

0xFA 0xFF 0x32 0x33 DATA[36] ROLL[4] PITCH[4] YAW[4] TIMESTAMP[2] CHECKSUM

where DATA[36] is raw accelerometer, gyroscope and magnetometer data.

The micro-controller validates the checksum of the above data and if the checksum is correct the pitch, roll and yaw values are updated to these new values received from the IMU. If the checksum is not correct the pitch, roll and yaw are not updated.

12.7 Implementing Remote Communication

Remote communication between the hexapod, the computer and the charging base is done by sending various different packets of data between these devices. A breakdown of each of the packets that can be sent or received by these devices is show below.

12.7.1 Hexapod Status Updates

A handshake is made with the computer in the form of a ping command sent to the computer and an expected response from the computer. If this ping fails for a set amount of time (default 50 seconds) the hexapod assumes to have lost connection to its host computer and stops moving to prevent damage (e.g. uncontrollably walking into danger).

Ping Command from Hexapod to PC

0xFF 0xFA

Ping Response from Computer to Hexapod

0x37 0x37 0x37 0x37 0x37 0x37 0x37 0x37 0x37 0x37

The hexapod also sends its current battery voltage and IMU data to the computer over various intervals (every 10 seconds for battery voltage and every 0.5 seconds for IMU data by default).

Battery Information from Hexapod to PC

A total of 10 bytes are sent to the PC as the base (which also uses this battery information when it is transmitted) is expecting to receive 10 bytes.

0xFF 0xF9 STATUS VOLTAGE[4] 0 0 0

where the STATUS is an integer from 0 to 3 representing either not charging, charging constant current, charging constant voltage or charging fault. The VOLTAGE is a float value represented by 4 bytes.

IMU Data from Hexapod to PC

A total of 26 bytes of data are sent to the PC.

0xFF 0xFE TIMESTAMP[2] PITCH[4] ROLL[4] YAW[4] SLOPEPITCH[4] SLOPEROLL[4]

where SLOPEPITCH and SLOPEROLL are the pitch and roll angles of the slope the hexapod is currently on. This is useful as the pitch and roll data will be set to a particular fixed, stabilised angle but the slope angle could be anything.

12.7.2 Hexapod Control

To control the hexapod, data is sent from the computer GUI containing information such as body height, walking direction and speed etc. These commands are shown below.

Movement Command

The computer sends 10 bytes of data for movement in the following format.

0x01 DIRECTION TWIST 0x02 SPEED 0x03 HEIGHT 0x04 FUNCTION CHECKSUM

where direction is an integer representing either forward, backwards, left, right, forward-right, forward-left, backwards-right or backwards-left. Twist is an angle by which the horizontal hip motors are offset to 'twist' the body of the hexapod without moving its foot positions. Speed is a percentage value of the maximum speed. Height is a percentage of a the maximum height. Function is an integer to represent various functions that the hexapod can perform. These include: stand the hexapod in its default position, turn clockwise or anti-clockwise by moving its feet, software reset the hexapod, start or stop charging and lift a leg so that the hexapod can walk onto the charging base.

Note that the hexapod responds to each movement command by sending back the following:

0xFF CHECKSUM

where CHECKSUM is a checksum of the data is received. If the computer GUI does not receive the correct checksum back after transmitting a movement command then it transmits that command again until it does receive the correct checksum. This is done so that a movement command does not get missed due to a momentary loss in communication.

Toggle Motors Command

0xFF 0xFA TOGGLE 0 0 0 0 0 0 0

where TOGGLE is either 1 or 0 to represent motors enabled or disabled.

Toggle Pitch and Roll Stabilisation

0xFF 0xFB TOGGLE CHANNEL

where TOGGLE is either 1 or 0 to represent enable or disable and CHANNEL is either 1 for pitch, 2 for roll or 3 for both.

Set PID Gains

Each PID gain is set by sending a separate command as the command to set all together would be larger than the 10 bytes expected by the hexapod.

0xFF 0xDE CHANNEL GAIN VALUE[4]

where CHANNEL is defined as above, GAIN is either 1 for K_p, 2 for K_i or 3 for K_d and VALUE is four bytes to represent the float value of the gain.

Toggle Charging

The hexapod receives a command from the PC to either start or stop charging, it then checks its current battery voltage before choosing to either start or stop charging by sending a command to the charging base.

If the battery is already fully charged it will not send the start charge command to the base.

0xFF 0xF8 0x37 TOGGLE 0 0 0 0 0 0

where TOGGLE is either 1 or 0 to start or stop charging respectively.

Reset Hexapod

The reset command comes from the movement commands. It sets the direction to 0 so that the hexapod stops moving and the function to 55 as below.

0x01 0x00 TWIST 0x02 SPEED 0x03 HEIGHT 0x04 0x37 CHECKSUM

12.7.3 Charging Base Control

After the hexapod has checked its battery voltage and deems it appropriate to charge it sends the following command which the base receives.

0xFF 0xF8 TOGGLE 0 0 0 0 0 0 0

where TOGGLE is either 5 for start charging or 6 for stop charging.

During charging the hexapod also checks if it currently is charging versus a boolean that stores whether or not it currently should be charging. If it should not be charging and it is charging then it sends the stop charging command to the base until it stops charging. If it is not charging and it should be charging then it sends the start charging command to the base until it starts charging.

12.8 Explanation Of The Full Code Process

The flow of the code is explained below. Block diagrams are used where possible but at times block diagrams were deemed to be too large and therefore explanations were used instead.

12.8.1 Initialisation Code

The code process that is executed once at startup is shown in a block diagram below.

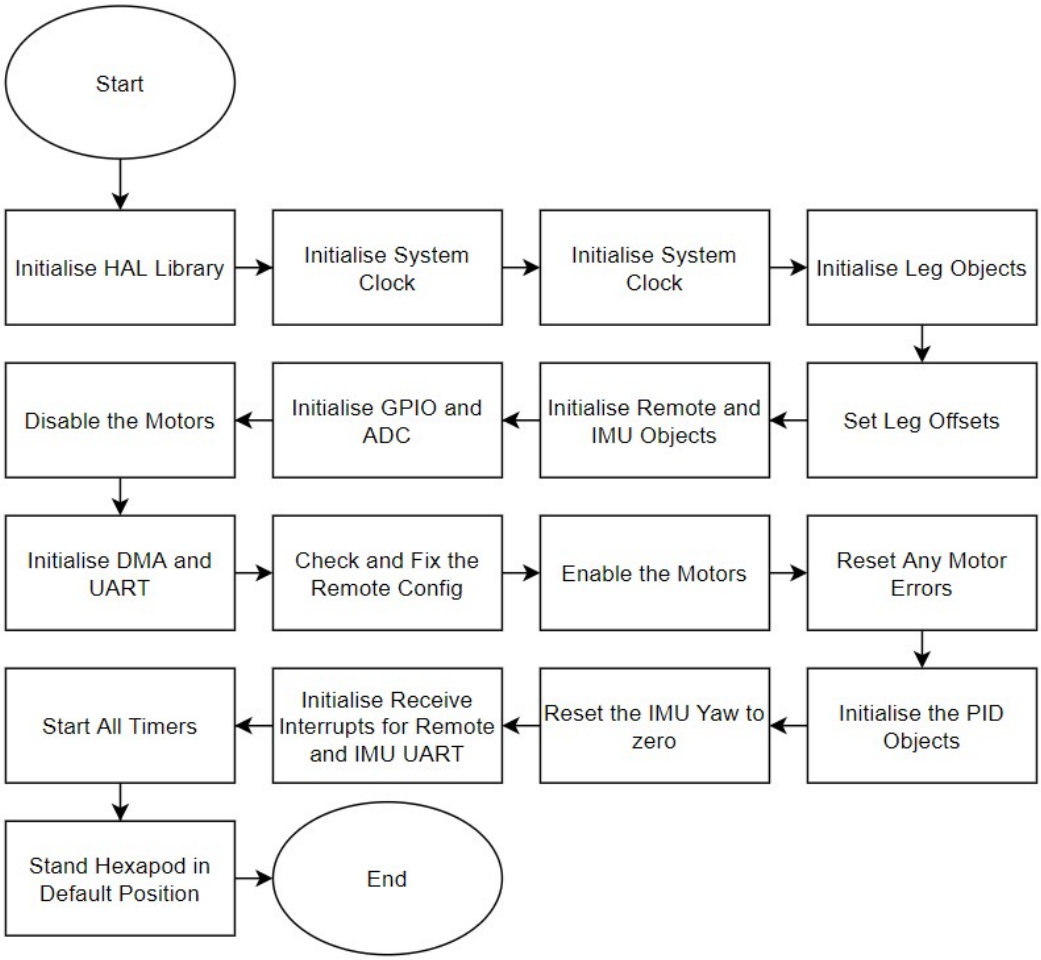


Figure 12.1: Hexapod Initialisation Block Diagram

12.8.2 Main Loop Code

The following code is executed continuously in a while loop and therefore does not contain the 'END' block.

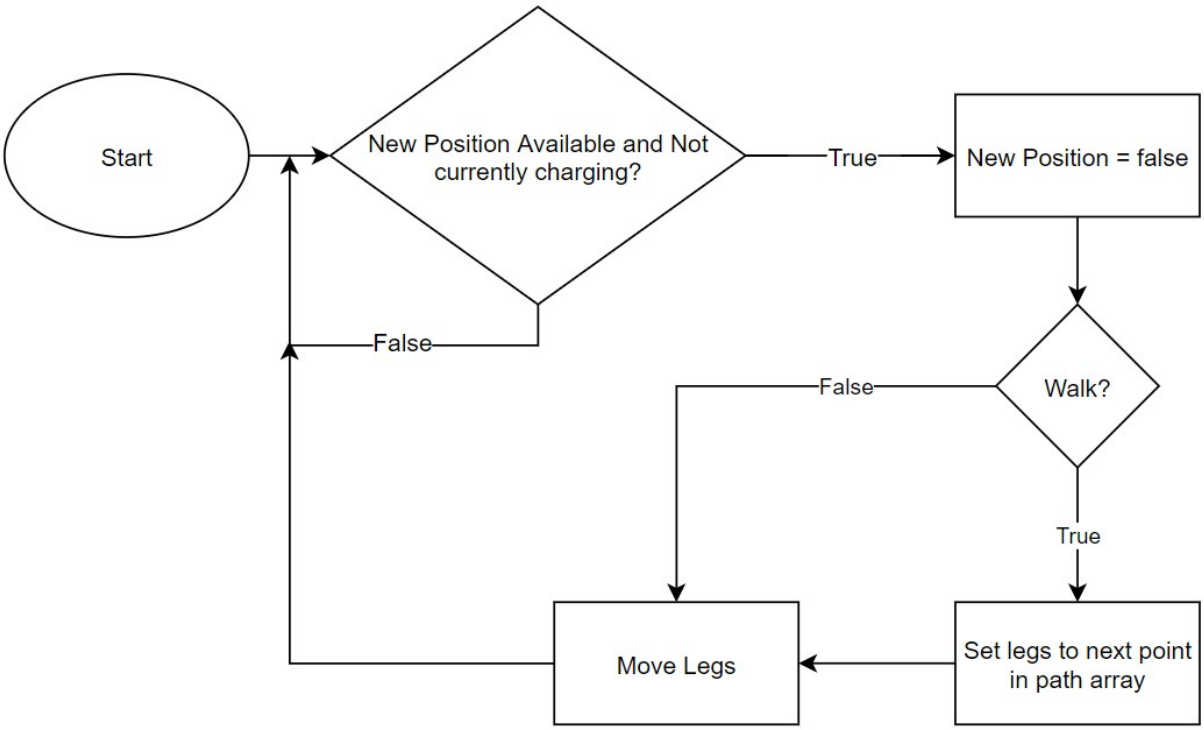


Figure 12.2: Hexapod Main Loop Block Diagram

12.8.3 Timer 2 Interrupt Code

This is the code that is executed in the Timer 2 interrupt function. The code in this interrupt is responsible for moving the hexapod's motors goal positions to their respective position during walking. It cycles through the generated path points for the current direction of motion.

The process of this interrupt can be seen in the block diagram below.

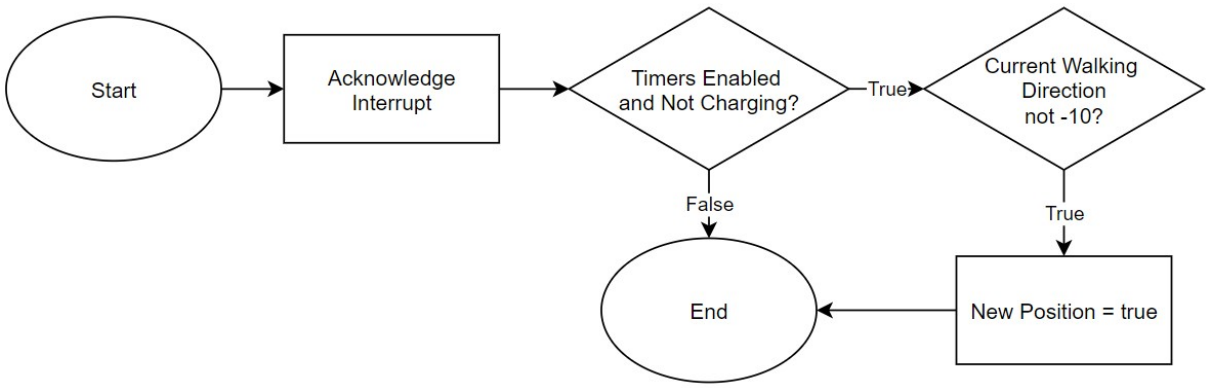


Figure 12.3: Hexapod Timer 2 Interrupt Block Diagram

Note that the interval of this timer is adjusted as the speed of walking changes. The timer interval is calculated based on the current speed as shown below.

Timer 2 Interval $= 600(1.1 - S)$.

where S is the speed percentage from 0 to 1.

This equation was determined experimentally to achieve smooth motion.

12.8.4 Timer 3 Interrupt Code

This is the code that is executed every 100ms in the Timer 3 interrupt function. The code in this interrupt has been broken down into different sections with each section running on every nth interrupt. This essentially allows different code to be run with different timing intervals using only one timer.

This is achieved by using a timer interrupt counter which increments on every interrupt. By comparing the modulus of this counter to a pre-defined constant a block of code can be run on every nth interrupt.

The longest timed block in this interrupt is run every 10 seconds. This function checks the current battery voltage of the hexapod by getting the ADC reading from the charging board. This process is broken down below.

- Set the enable battery voltage read pin to HIGH.
- Begin ADC conversion.
- Poll for ADC conversion (wait until the conversion is complete).
- Set the enable battery voltage read pin to LOW.
- Read the two GPIO status pins from the charging board.
- From the two status pins determine the current state of the batteries.
- Return a charging status enum that contains the battery status and voltage.
- Transmit this information through the remote to the charging base and the computer at which point the computer displays this information to the user.

The pin connected to the ADC that is used to read the battery voltage is connected through a digital switch. When the switch is turned on the batteries are connected to the ADC pin (through a voltage divider to adjust the battery voltage to 0 - 3.3V) and when the switch is turned off the batteries are disconnected. This was done so that when the hexapod is not powered up the battery is not connected to the micro-controller and therefore the micro cannot be powered erroneously through this connection. Albeit a small, if not negligible, this switch also aids in preventing unwanted power consumption caused by constantly having the battery connected to the micro-controller.

The next timed block is executed every 5 seconds. This is where the ping command is sent to the computer and the micro-controller checks if it has received a response. It keeps count of each time it does not receive a response and if 10 responses are missed in a row it assumes that there is a loss of communication and stops walking.

The next timed block is executed every 500ms. This sends the current IMU data to the computer.

The next timed block is executed on every interrupt iteration and therefore every 100ms. This block simply sends the request data command to the IMU so that the IMU can then respond with the orientation data of the hexapod to keep its current attitude up to date.

12.8.5 UART/DMA Transmit Complete Interrupt Code

This is the code that is executed after every transmission of serial data to either UART 1, 2 or 3.

- Clear the transmission complete interrupt flag

12.8.6 UART/DMA Receive Complete Interrupt Code

This is the code that is executed every time either UART 2 or 3 has finished receiving serial data.

If the data received came through on UART 1 (the motors) nothing is executed. At the point of completion of reception the motor control tables in the motor objects will be populated with the latest data from each motor. As such if this data is to be used anywhere at this point it is already up to date and no further processing is required.

If the data received came through UART 2 (the remote) the following steps are taken.

- The remote data is processed to extract all movement parameters or functions being commanded to perform.
- The hexapod's parameters are updated with these values and any commanded functions are performed.
- New leg paths are computed with the updated movement parameters.

If the data received came through UART 3 (the IMU) the following steps are taken.

- The data is processes to extract the timestamp, pitch, roll and yaw data and convert them each into their respective float values
- The PID control loop for both pitch and roll is updated with the new IMU data (if the pitch and roll stabilisation is enabled)
- A limit is applied to the output of the PID controller to prevent the hexapod from angling itself too much and causing damage (default angle limit is 34 degrees, found through experimentation).
- The hexapods attitude is updated with the output of the PID controller and the new position boolean is set to true

12.8.7 UART/DMA Interrupt Handlers

The UART and DMA interrupt handlers are called on the transmission and reception of serial data. They are called after each byte and not after all data has been received and for this reason they are not used to process any serial data. They simply acknowledge their respective interrupt to allow the next byte to be receive. Without these interrupt handlers, and without each one acknowledging its interrupt, the serial data communication fails after transmitting or receiving one byte and the remaining serial data is lost.

12.8.8 Systick Interrupt Handler

The Systick handler is responsible for incrementing the system code clock tick counter on each clock tick. This is its only function and no other code is to be executed within this interrupt handler.

12.9 Extern ”C”

All the interrupt handlers for the STM32F4 are designed to be compiled and executed in C as opposed to C++. This created a problem of combining the C++ object orientated hexapod control code with the C code for these handlers. The command ”extern C” is applied around these interrupt handlers which tells the compiler that any code with the ”extern C” block must be treated as C code and not C++ code [67].

12.10 Controller Graphic User Interface

A control graphic user interface (GUI) was developed to allow for communication between a computer and the hexapod. This program was developed using java and Netbeans [68] because Netbeans allows for fast development of GUIs using its drag and drop GUI builder.

The software was designed to interface with a joystick and the remote transceiver connected to the computer to allow for position control of the hexapod. The commands that the software can send to the hexapod are listed below.

1. Move in one of eight directions (forward, left, diagonal etc).
2. Adjust the height of the hexapod (up or down).
3. Adjust the speed of the hexapod (faster or slower).
4. Stand the hexapod in its default, neutral position.
5. Twist the hexapod body either clockwise or counter-clockwise without moving its legs.
6. Turn the body of the hexapod either clockwise or counter-clockwise by moving its legs.
7. Lift a leg of the hexapod and adjust it to the correct height to be able to walk onto the charging base.
8. Toggle the motors on or off.
9. Toggle the pitch and roll stabilisation on or off.
10. Set the PID parameters of the pitch and roll stabilisation.
11. Software reset the hexapod's micro-controller.
12. Start or stop charging of the hexapod.

When the hexapod starts up it pings the computer and the computer responds, setting up a connection with the computer. The hexapod pings the computer continuously during operation and if the computer does not receive this ping or hexapod does not get a response from the computer, the system warns the user and the hexapod ceases operation after a timeout. This is to prevent unexpected or undesired motion of the hexapod.

The hexapod sends constant data to the computer to show its current state as well as any errors that have occurred. Battery status information such as the current voltage of the battery and whether or not the battery is currently charging and the phase of charging the battery is in, whether its constant current charging or constant voltage charging.

A screenshot of the GUI can be seen in figure 12.4 below.

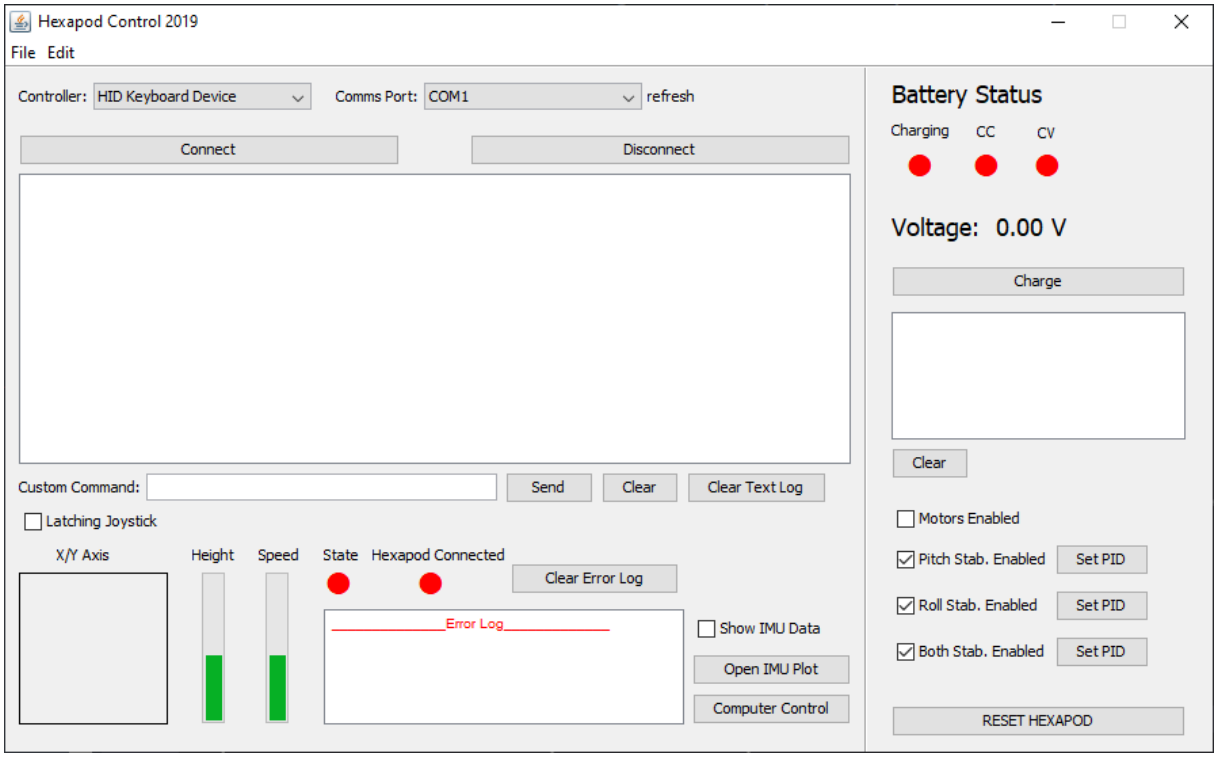


Figure 12.4: Hexapod Control GUI

The IMU data is transmitted and plotted in real time. This includes pitch and roll angle of the slope the hexapod is currently on and the true pitch and roll of the hexapod relative to a flat ground. A screenshot of a plot of the pitch and roll data can be seen in figure 12.5 below.

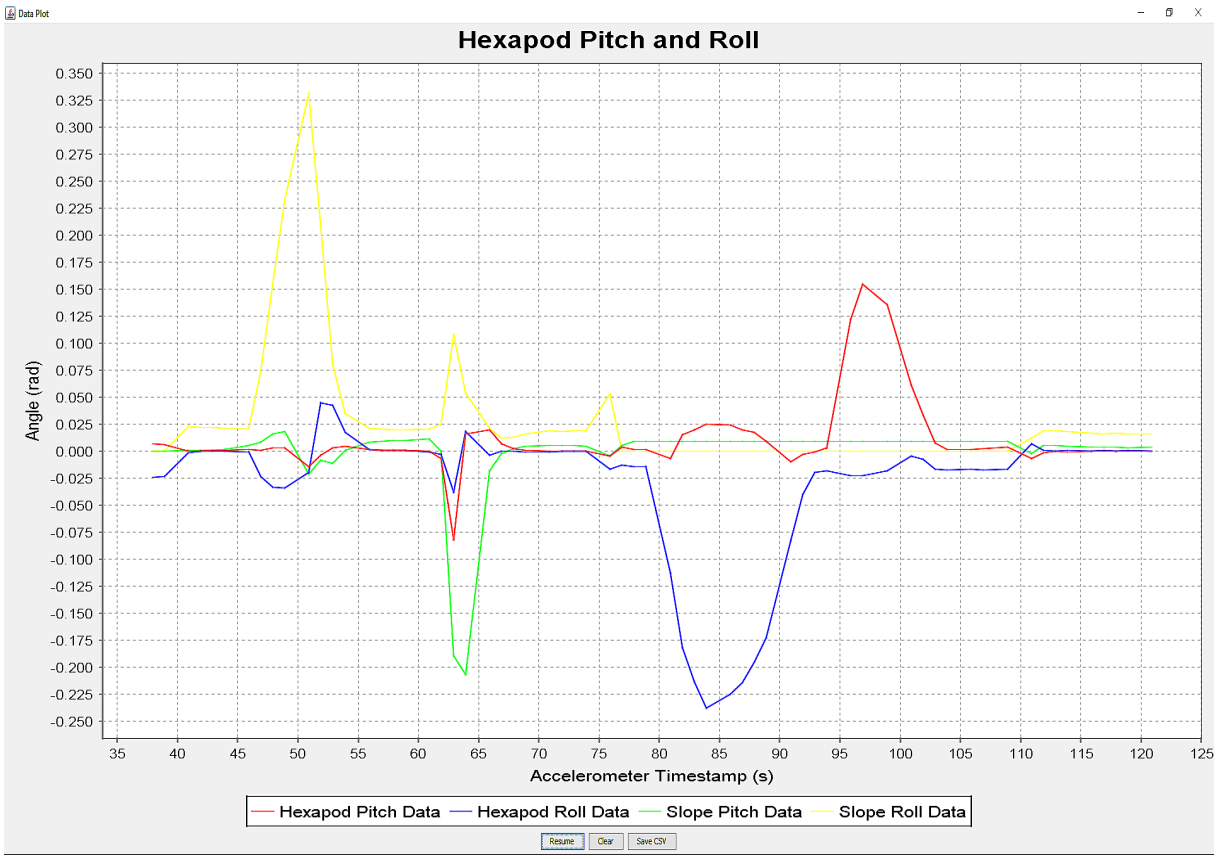


Figure 12.5: Hexapod Control GUI Real-time IMU Plot

12.11 Discussion of Programming

The programming of the hexapod was successfully implemented on the STM32F4 micro-controller using C++. Functionality was added to allow for remote communication with a control GUI running in java on a computer. This allowed for the successful monitoring of the hexapod's state (battery voltage and any errors that may have occurred).

The C++ program was designed to be object orientated so that it is modular and further improvements can be easily made either to individual objects or to the control system as a whole.

Links to a GitHub repository containing both the micro-controller code are found in Appendix C.

Included in this repository is a readme containing information on every object and function in the micro-controller code.

13 Hexapod Charging Base and Battery System

The hexapod charging base and circuitry was designed by Sven Weihe for his undergraduate thesis [7].

The work in this project included the development of the necessary C++ code to facilitate communication with this charging base. When the battery of the hexapod is below a certain threshold, and the hexapod is placed onto the charging base, the hexapod sends a start charging command to the base remotely. The base then powers up and begins charging the batteries wirelessly.

During the charging process the hexapod monitors the current charging state and battery voltage through the charging circuit board. The hexapod then transmits this information to the control GUI so that the user can view the current battery voltage as well as the charging phase which can either be constant current or constant voltage. This information allows the user to safely charge the hexapod.

The battery system was also developed in Sven's project. It was made up from six cylindrical 18650 lithium-ion cells in series with one another. This provides a supply voltage of 25.2V.

A switch mode power supply was designed to regulate the battery voltage to the 15V required for the motors however it did not function as it was intended and eventually a store bought power supply replaced it on the hexapod.

More detail about the individual circuits designed for wireless charging and battery management system can be found in [7]. The wireless charging base can be seen in the figure below.

Figure 13.1: Hexapod Charging Base [7]

14 Hexapod Testing and Verification

The hexapod with all of its components active was tested to verify that the walking speed and distances were correct as well as the remote control communication and the IMU stabilisation. Each component was tested separately at first to verify each one works and then all components were tested as they operated together.

14.1 Walking Test

A walking test was performed over flat ground with the IMU pitch and roll stabilisation disabled. This was done to verify that the hexapod was walking correctly in the direction it was meant to be moving at its desired speed. Distances were marked out in the Robotics and Agents Research Lab and the hexapod walked these known distances. Straight lines were added to the track to determine how straight the hexapod was walking. This test was simply done by observing the hexapod as it walked.

The hexapod can be seen on the test track in figure 14.1 below. This photo was taken at the beginning of the test path.

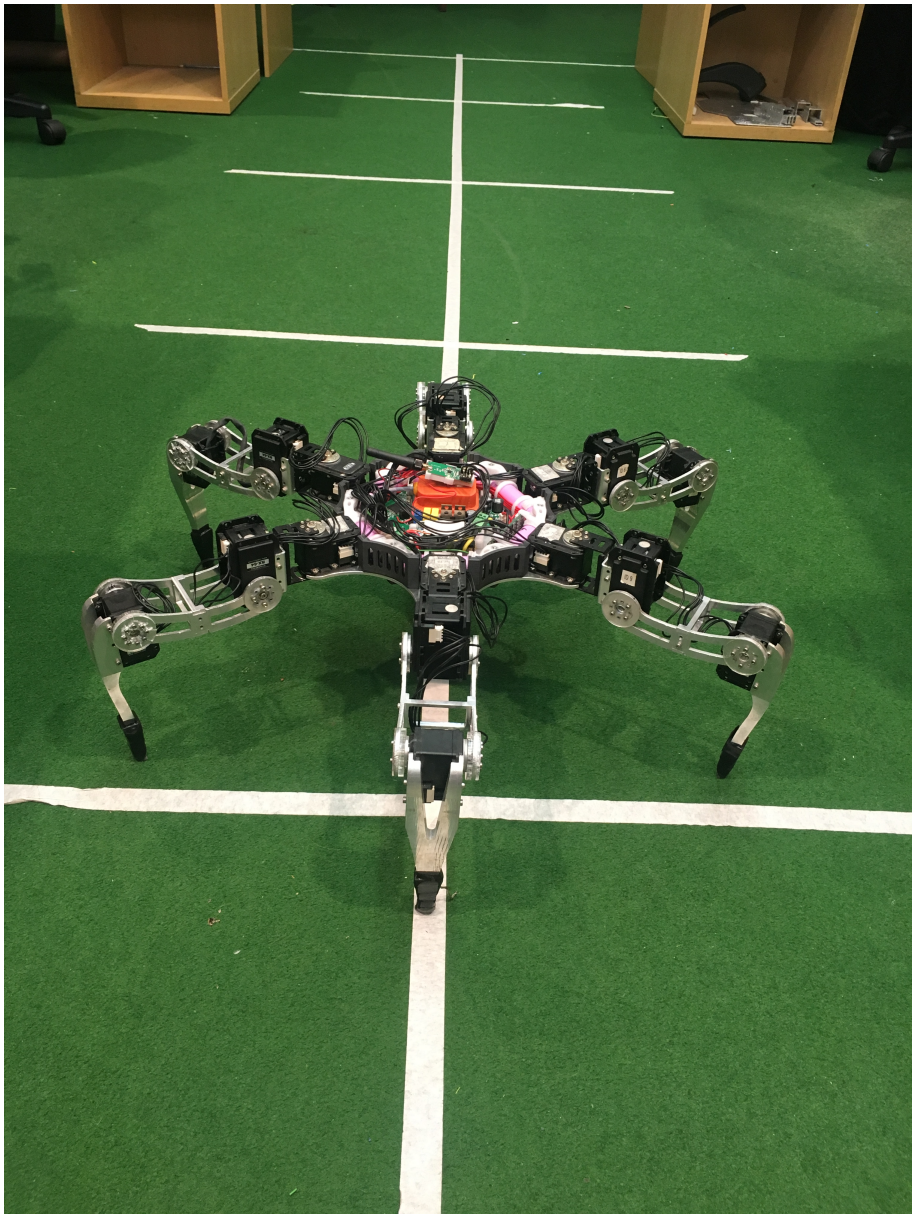


Figure 14.1: Hexapod Walking Test Track at Start

The following figure shows the hexapod at the end of the test track.

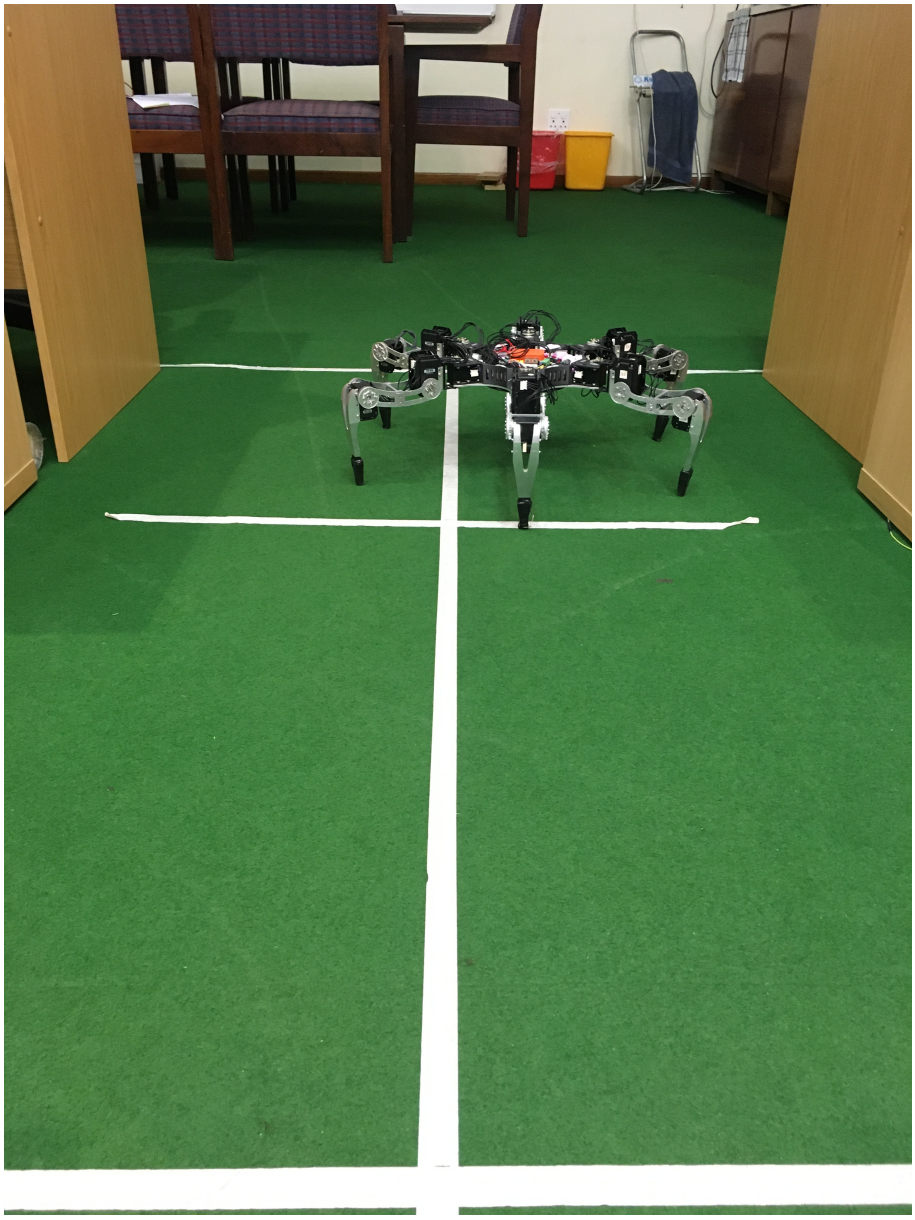


Figure 14.2: Hexapod Walking Test Track at End

It can be seen from the figure above that the hexapod drifts slightly off the straight line path due to the slipping of its feet. If a foot slips the hexapod gets pushed in the direction of that foot by the other feet, causing it to veer off the track.

14.1.1 Distance Test

The hexapod counts each leg stride and keeps a record of this value. This is equal to the total distance that the hexapod has walked since its most recent reset. This value is to be compared to the measured out distance to determine if the distance control is correct.

Testing was performed in each direction that the hexapod can walk, over varying distances as shown in the table below.

Table 14.1: Hexapod Walking Distance Test Results (Strides)

Forward	Backward	Left	Right	Commanded Distance(Strides)
21	20	22	21	20
41	43	41	44	40
64	66	65	63	60

Each stride for this test was set to be 50mm in length. Using this value the number of strides performed was converted into the distance that the hexapod walked. This was then compared to the distance the hexapod was expected to walk.

This lead to an average error for each direction as well as each distance as shown below.

Table 14.2: Hexapod Walking Distance Test Results (Error)

Forward (mm)	Backward (mm)	Left (mm)	Right (mm)	Commanded Distance (mm)	Average Error (mm)
50	0	100	50	1000	50
50	150	50	200	2000	112.5
200	300	250	150	3000	225

It can be seen from the results above that during each test the hexapod stepped more times than it should have over the distance it walked. This was visibly due to the the feet slipping on the floor while it walked which lead the hexapod to step between two and three steps more than required, on average. The increase in error as the distance gets larger confirms this because as the distance increases there is more time for the hexapod to slip and lose steps, adding to the number of steps needed to cover the distance and therefore increasing the error in steps.

14.1.2 Speed Test

The speed test was done by walking the hexapod across the track while measuring the time it takes to reach each marker line. From this information the walking speed of the hexapod can be calculated.

The commanded walking speed is calculated from the known time that one stride will take which will vary at different speeds.

Table 14.3: Hexapod Walking Speed Test Results

Forward (mm/s)	Backward (mm/s)	Left (mm/s)	Right (mm/s)	Commanded Speed (mm/s)
81.23	70.27	75.07	77.04	73.26
85.94	70.27	79.08	80.45	73.26
75.93	69.91	71.99	71.11	73.26

The above test lead to the determination of the error in hexapod speed control. The average error in speed was calculated to be 2.43mm/s with a standard deviation of 2.73mm/s.

14.2 IMU Stabilisation Test

To test the IMU stabilization response, the hexapod was placed on the center of a desk while the desk was tilted to various angles and the pitch and roll response of the hexapod monitored through the GUI pitch and roll plot. The plot shows the angle of the slope of the desk as well as the angle of the body of the hexapod.

The pitch and roll plots for varying slope angles can be seen below.

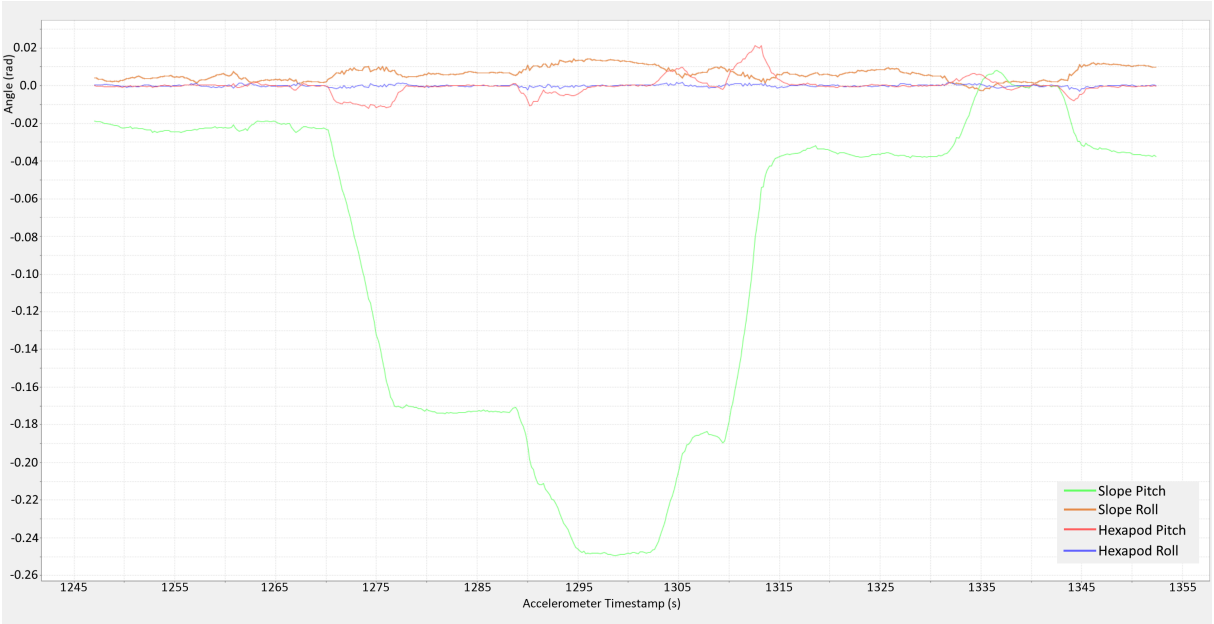


Figure 14.3: Hexapod Stabilization Pitch Test

In the above plot, the green line shows the true pitch of the slope while the red line shows the pitch of the hexapod. It can be seen that as the true slope pitch decreases to -0.26 radians, the hexapod pitch remains near constant at ± 0.01 radians. This shows that the pitch control of the hexapod is successfully keeping the hexapod body stable.

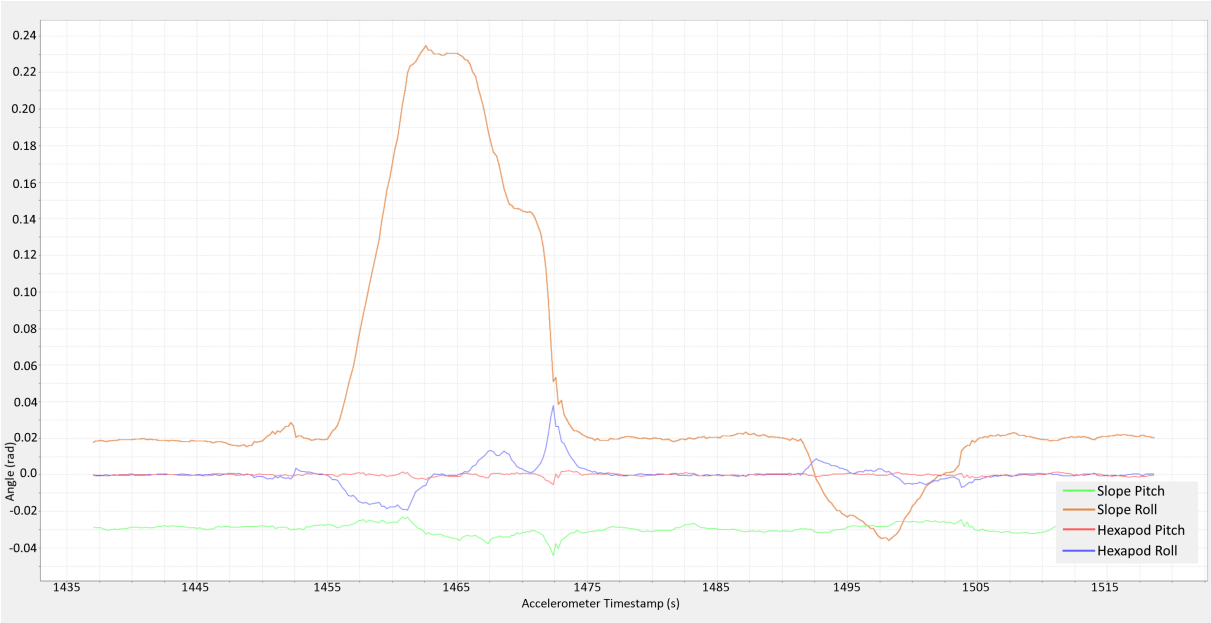


Figure 14.4: Hexapod Stabilization Roll Test

In the above plot, the orange line shows the true roll of the slope while the blue line shows the roll of the hexapod. It can be seen that as the true slope roll increase to ± 0.24 radians, the hexapod roll remains near constant at ± 0.02 radians. This shows that the roll control of the hexapod is successfully keeping the hexapod body stable.

Figure 14.5: Hexapod Stabilization Pitch and Roll Test

The above plot shows that both the pitch and roll of the hexapod are remaining constant at the same time. In this test, the true slope pitch and roll (green and orange lines respectively) are changed between ± 0.2 radians at the same time it can be seen that the hexapod pitch and roll (red and blue lines respectively) both remain constant at ± 0.02 radians.

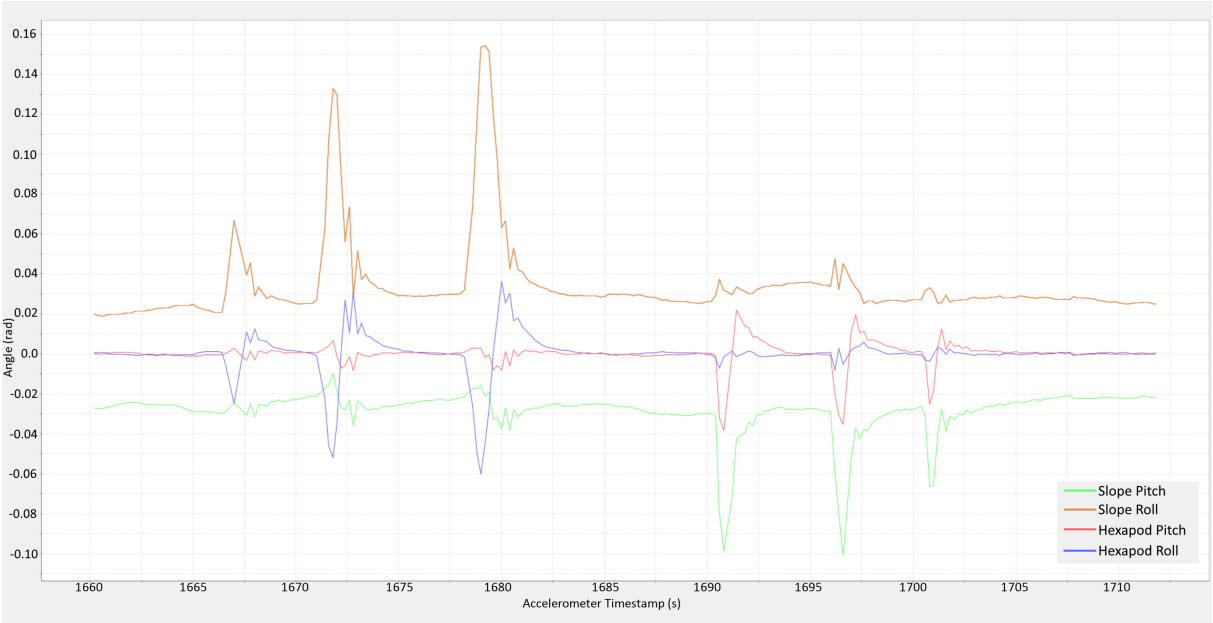


Figure 14.6: Hexapod Stabilization Fast Response Pitch and Roll Test

A final stabilization test to verify the speed of the response of the stabilization shows that with fast spikes in true slope pitch and roll, the hexapod pitch and roll follows these spikes with a significantly reduce magnitude (± 0.1 radians decrease). The hexapod's pitch and roll is corrected back to zero with some overshoot.

14.3 Full Test

A full test was performed to verify that the hexapod walks correctly over uneven terrain while maintaining a level body. This was done by walking the hexapod over a platform that was tilted such that there was an uphill side and a downhill side to the test. The hexapod walked over this slope and the IMU data was captured to confirm that it remained level. The setup for the test can be seen in figure 14.7 below.

The test began with a slope of ± 0.1 radians and was repeated a number of times with varying slope angles. The upper limit of the slope angle of 0.6 radians was selected as it is the software limit of the stabilisation of the hexapod. This limit was determined experimentally and is set to avoid damage to the hexapod.

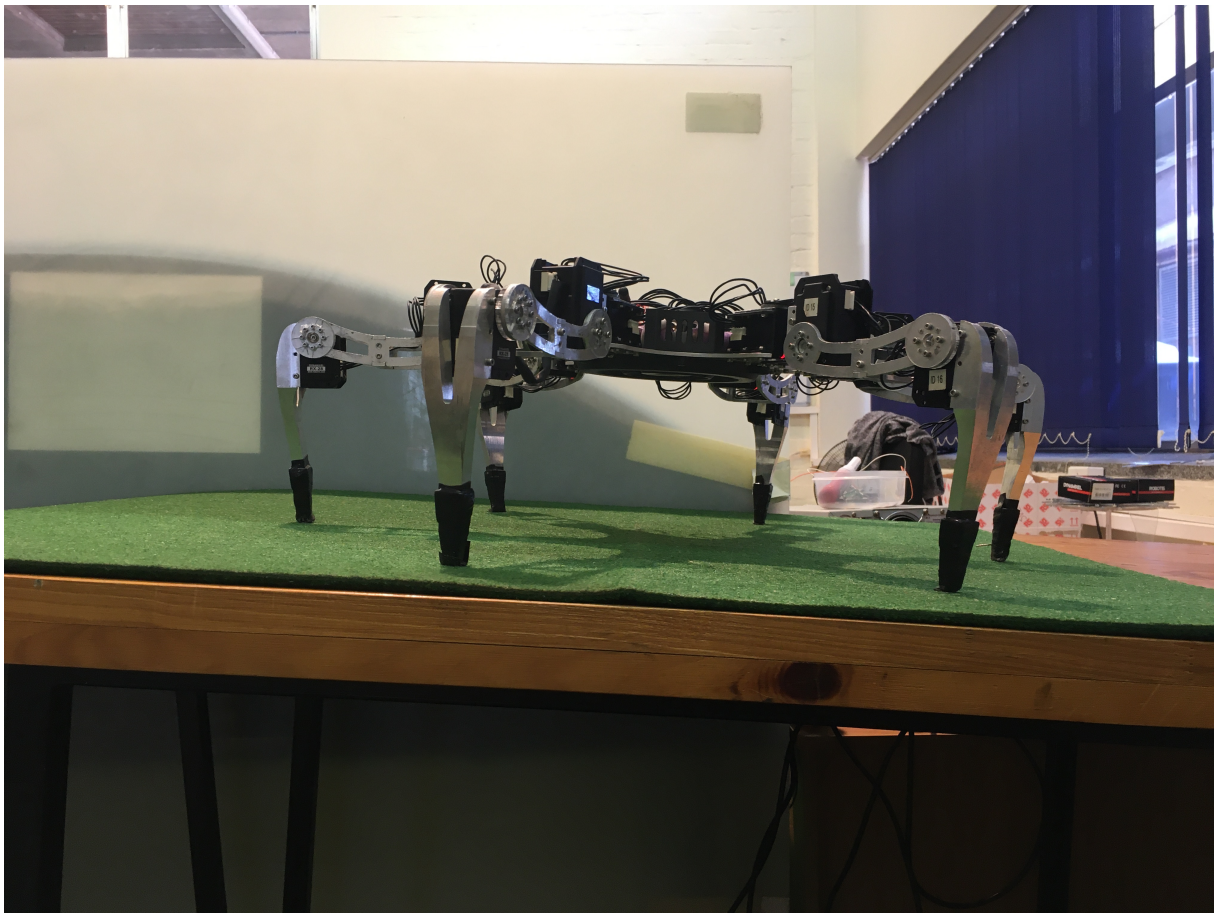


Figure 14.7: Hexapod Full Walking Test

First the hexapod walked over the slope without any stabilisation to get a base line. The IMU data plot for this test is shown below.

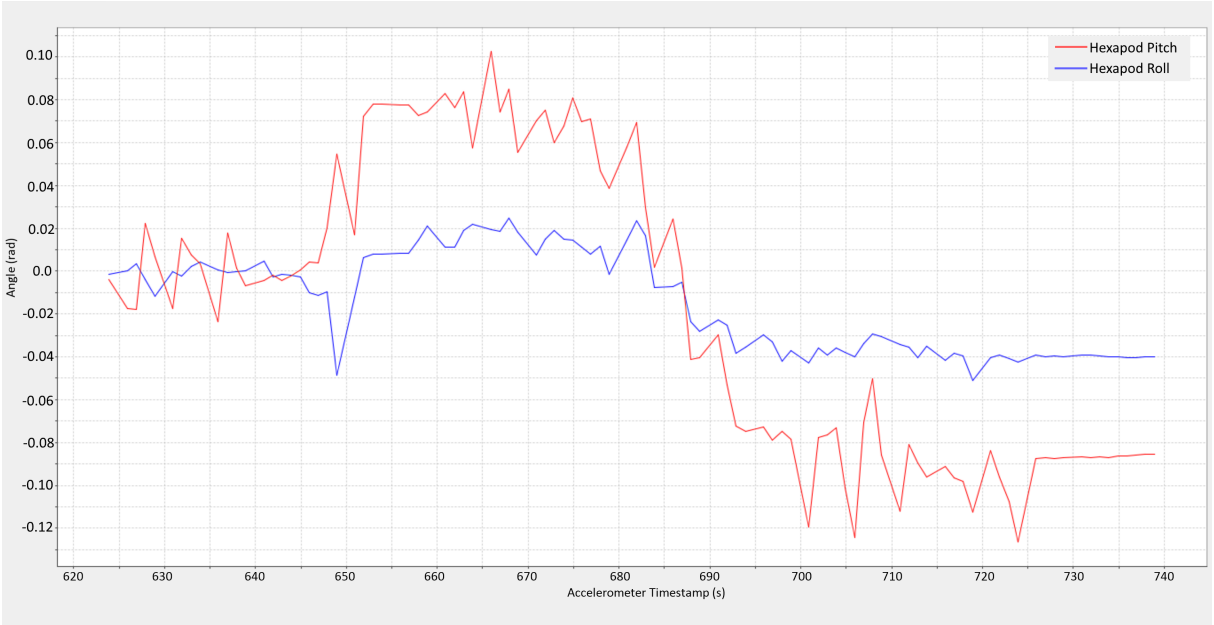


Figure 14.8: Hexapod Full Walking Test with Stabilisation Off

The stabilisation was then turned on and the test was performed again. The IMU data plot for this test is shown below.

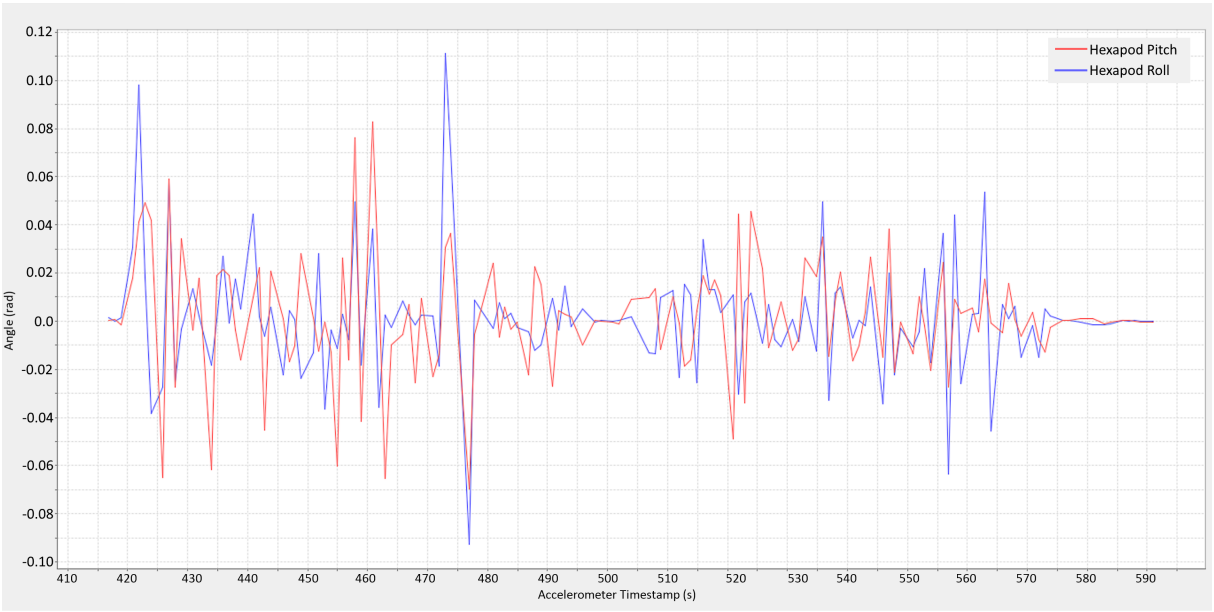


Figure 14.9: Hexapod Full Walking Test with Stabilisation On

The plots above show that the stabilisation of the hexapod while walking over uneven terrain functions correctly although it is not smooth. Figure 14.7 when the hexapod is standing still with stabilisation off can be compared to figure 14.10 below where stabilisation is switched on.

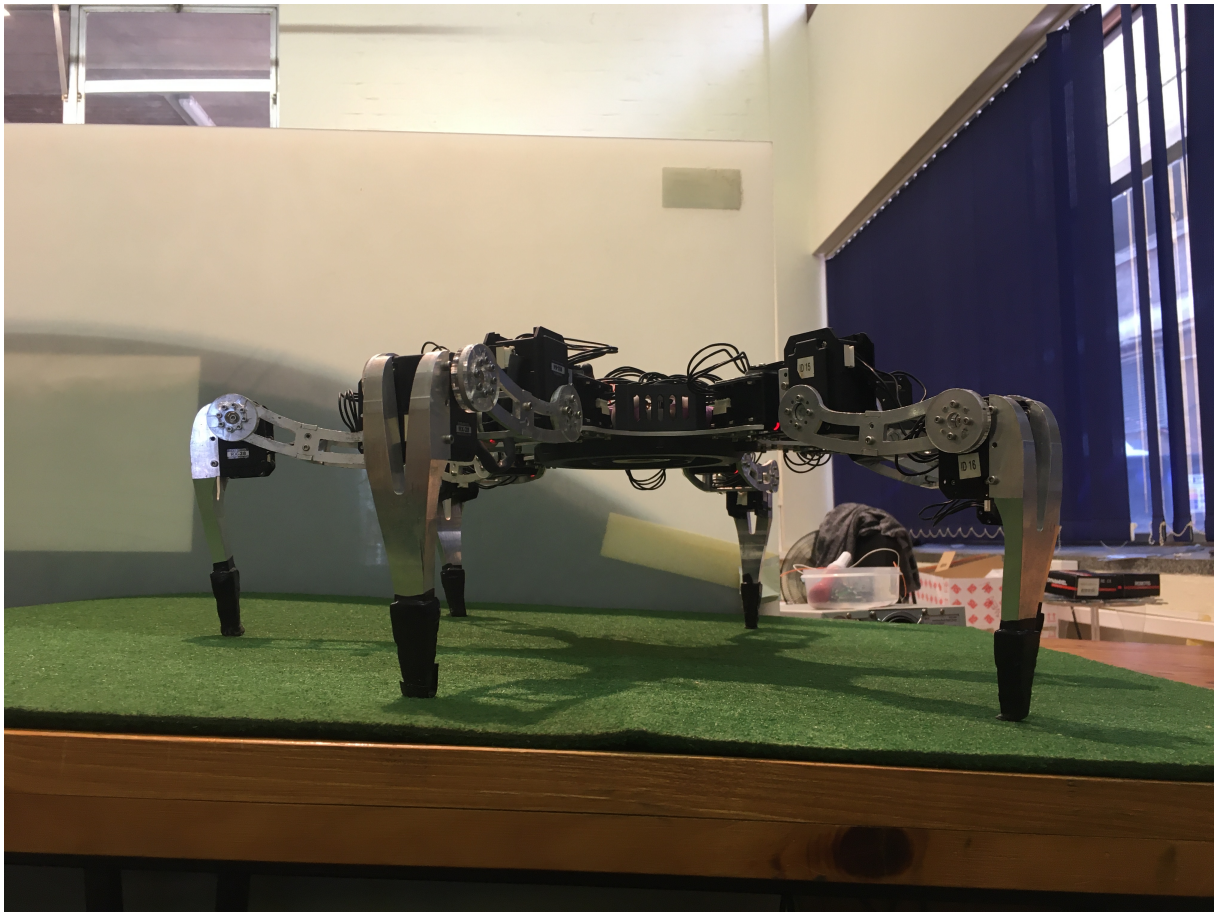


Figure 14.10: Hexapod with Stabilisation On

14.4 Discussion of Results

The tests performed above successfully verified the functionality of the hexapod robot. Position and speed tests were performed to verify the path generation algorithm and these tests showed that the position control of the hexapod as a whole is accurate to $\pm 6.0\%$ on average while the speed control of the hexapod is accurate to $\pm 3.3\%$.

In both of the above tests there was slipping between the feet of the hexapod and ground on which it walked. This contributed to the errors seen. However during normal operation of the hexapod, the accuracy of the position and speed of the hexapod isn't crucial as the hexapod is to be controlled by a user who can choose to walk the hexapod for longer, if slipping is noticed, in order for it to reach its destination correctly.

A stabilisation test was performed and various IMU plots were generated. These tests show that the hexapod stabilises its body using the tuned gains of the PID controller. Without the PID controller turned on the hexapod's body is simply parallel to the slope on which it stands however with the stabilisation turned on the hexapod's body's pitch and roll angles are always zero. A software limit on the stabilisation of 0.6 radians for pitch and roll was implemented to prevent mechanical damage to the hexapod.

A complete stabilised walking test was performed and yielded results that show an improvement of attitude of the hexapod body while walking over uneven terrain. Its attitude is centered around the zero line on the plot, however due to the motion of walking, it fluctuates around zero.

Through the repetition of the complete walking tests at varying slope angles it was determined that this fluctuating motion around the zero line during walking did not increase as the slope increased. This shows that the walking stabilisation works correctly.

15 Final Hexapod

The final hexapod can be seen in the figures below. This includes the battery system circuitry and the wireless charging base designed by [7] in 2019.

The figure below shows the final layout of the circuitry of the hexapod (note that the motor cables are not plugged into the control PCB so that the circuit boards can be seen more easily).

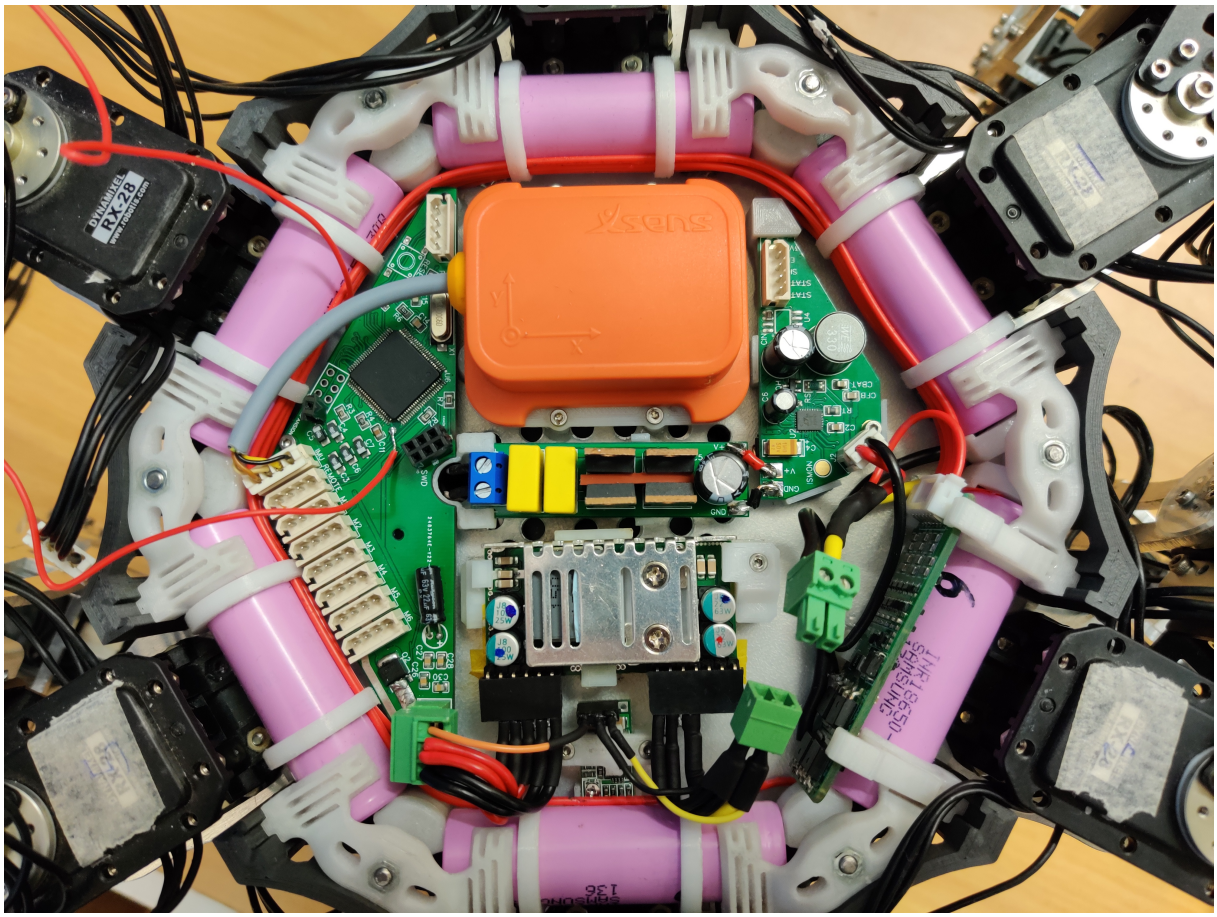


Figure 15.1: Completed Hexapod Circuit Layout

The figure below shows the final hexapod with the top plate attached. The hexapod is situated on the wireless charging base.

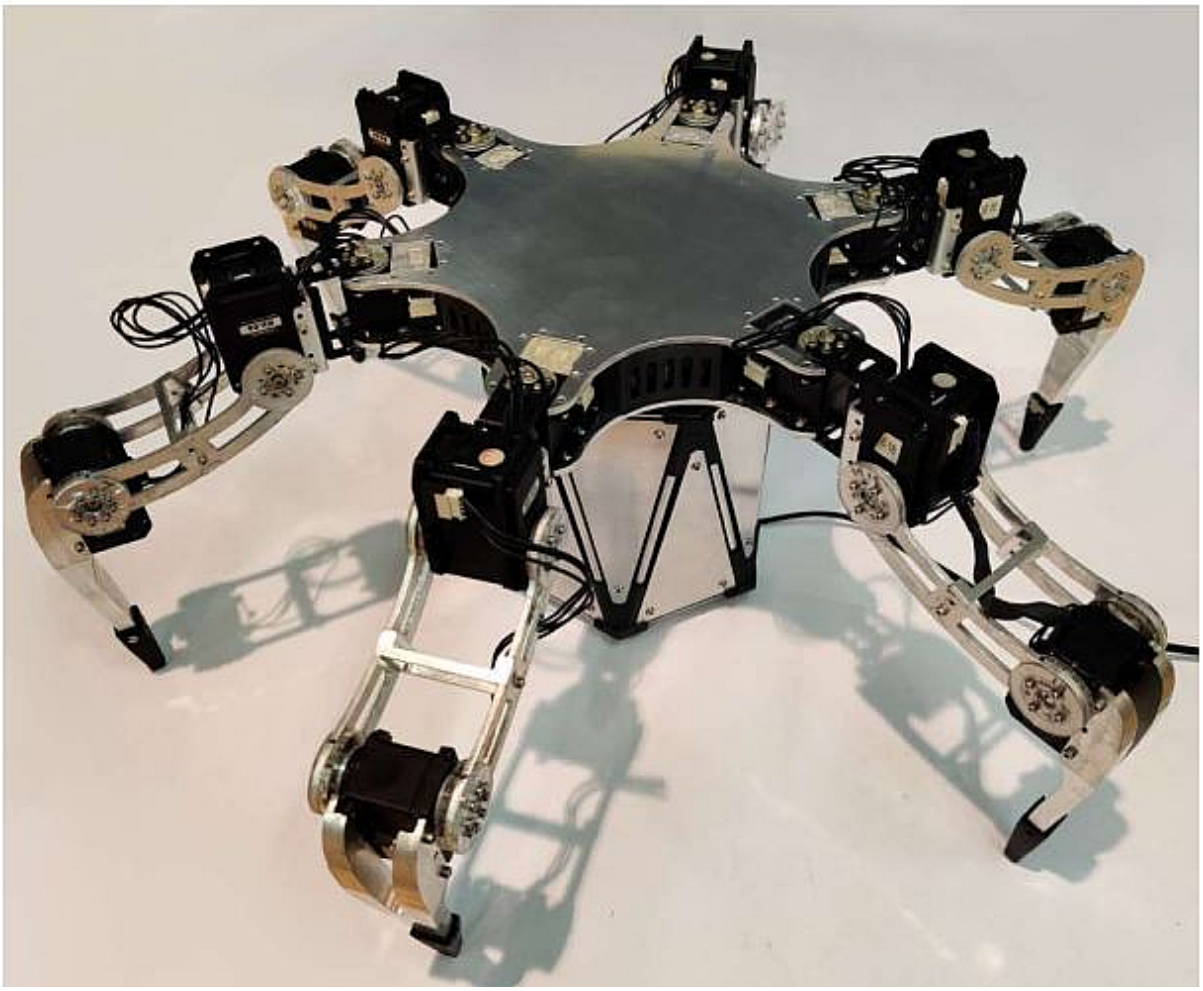


Figure 15.2: Completed Hexapod on Charging Base

The figure below shows the hexapod using the PID stabilisation to keep its body level while standing on the test slope.

Figure 15.3: Hexapod Stabilised on a Slope

The figure below shows the hexapod with two of its legs on an object while its other legs remain on the ground. The PID stabilisation keeps the body of the hexapod level.

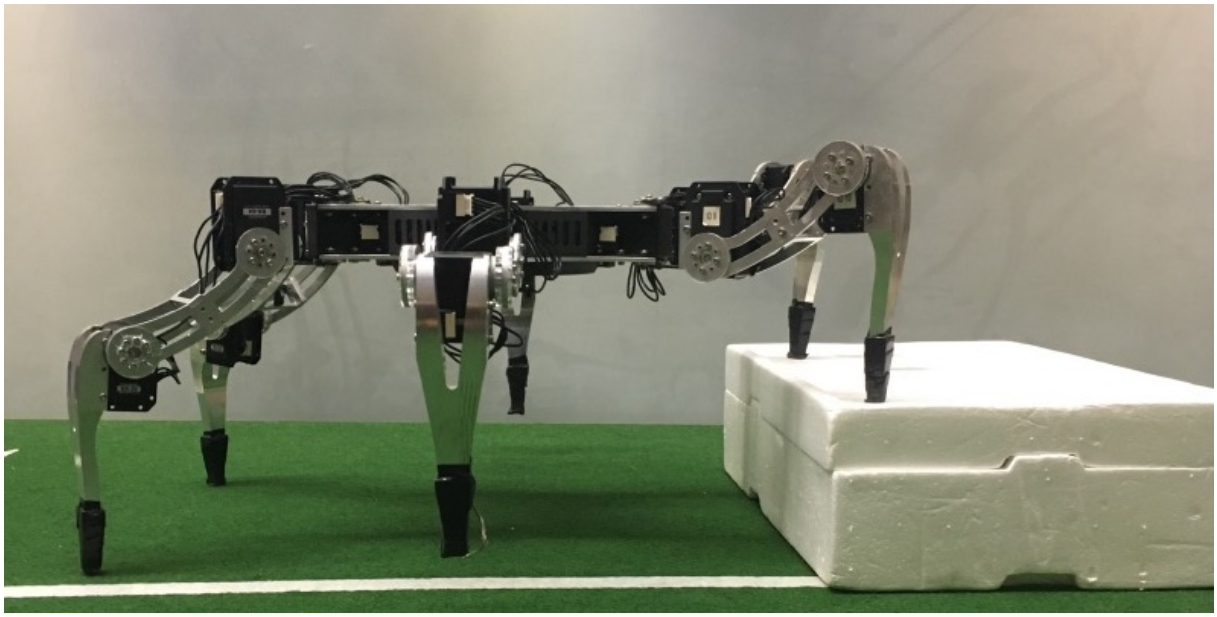


Figure 15.4: Hexapod Stabilised on a Box

15.1 Budget

A detailed budget was kept for each component purchased for the hexapod robot, not including the wireless charging base and circuitry designed by Sven [7]. This budget can be seen in Appendix D.

The budget shows that a total of R3 710.82 was spent on the hexapod during this project.

16 Conclusion

The hexapod robot at UCT's Robotics and Agents Research Lab has been under development for some time. At the start of 2018 it was controlled using a mobile computer running Windows 7 which led to it having a slow start-up time and the LabVIEW software running on it would often crash, requiring a reboot.

The aim of this project was to continue the work on this hexapod and improve its control system such that it can be used in a search and rescue environment. In search and rescue it is important to be able to respond quickly to a situation; if the robot that is assisting rescuers needs six minutes to start up, it is wasting crucial time. For this reason the controller of the hexapod was switched to an STM32 micro-controller.

The control of the hexapod MK1 was simply a brute force method of reading in motor positions from a file and moving to said positions. This does not allow for the generating or changing of the motion of the hexapod in real time. In order to generate a real time position control algorithm the mathematical model of the hexapod is required. The forward and inverse kinematics as well as the differential and inverse differential kinematics were developed to facilitate this. A dynamic model of the leg of the hexapod was developed to relate the foot forces to the motor torques, however the dynamics was not implemented on the final hexapod due to the slow read time of the Dynamixel motors.

The mathematics developed was implemented onto the hexapod and tests were performed to verify the functionality of it. These tests concluded that the position control of an individual leg was accurate to 1.7mm while the speed control was accurate to 2.4mm/s.

A search and rescue robot must have sensors such as cameras on-board. A camera mounted onto a robot is only useful if it can be pointed in a particular direction and kept facing that direction. For this reason a PID controller was developed to stabilise the body of the hexapod using pitch and roll data from an IMU mounted to the hexapod.

Leg motion planning algorithms were developed which comprised of both a foot position planner and a foot trajectory planner. The foot position planner used parameters such as body height and stride length to determine start, middle and end points for the motion of the foot. These values were passed into the trajectory planner which uses a single sixth order polynomial to generate the foot path moving through the start, middle and end points. It was seen that the single sixth order polynomial out performed lower order polynomials as well as only requiring one equation to represent the full swing phase.

A wireless radio frequency remote control was implemented on the hexapod. This allowed a computer run GUI to communicate remotely with the hexapod to send it commands such as movement commands and to receive status information from the hexapod such as battery voltage and IMU pitch and roll data.

Complete hexapod walking tests were performed to verify the leg motion planning algorithms as well as the PID controller for stabilisation. These test showed that the hexapod walked successfully on a slope while maintaining a level body, however the body pitch and roll was not smooth. This was due to the walking motion of the robot.

It is noted that the ”effectiveness” and ”smoothness” of the walking motion is difficult to quantify. The degree to which the algorithm is effective was judged based on the visual performance increase when compared to the previous MK1 hexapod. A further study to verify the true effectiveness of the walking motion is recommended.

It is noted that due to the slow read time of the motors, the hexapod is not able to determine whether or not its legs are actually touching the ground when they should be. If the micro-controller could read all motors at the same time then the motor torque information could be used to determine which feet are on the ground and which are not.

The PID controller adjusts the height of all legs to achieve a desired pitch and roll of the body of the hexapod, however if a leg is in fact not touching the ground the PID controller does not take this into account and as such the motion during walking is not always smooth as the PID controller is not functioning efficiently. Recommendations are later made to remedy this issue.

The hexapod is capable of walking using a path generation algorithm that allows it to be versatile and modular. The C++ developed code is split into classes which allows for further modularity. The mathematical models developed apply to this hexapod in particular, however they have been determined based on link length variables and can therefore be applied to any three degree-of-freedom hexapod robot with a similar anthropomorphic leg configuration.

The overall motion of the final hexapod is an improvement over the previous MK1 hexapod. Although the MK1 hexapod had more functionality such as a gripping mode and more gait options it did not have any degree of autonomy in the control system it used. The final hexapod generates all path points in real time and this is a significant improvement over the MK1 hexapod.

17 Recommendations

There are a number of recommendations for future work on the RARL hexapod. These recommendations are outlined below.

17.1 Gait Study

More work can be done on the hexapod to develop different gait profiles for varying terrain to further improve its search and rescue capabilities. These can include a gripping gait which will allow the hexapod to interact with objects around it. This can be done without any changes to the current mechanical structure of the hexapod simply by adding more gait options to the software controlling the hexapod.

17.2 Walking Algorithm Effectiveness

It is difficult to quantify the effectiveness of a walking algorithm, there are a number of parameters that can be used to judge this such as smoothness, base motion rejection, foot position vs hexapod position accuracy as well as foot speed vs hexapod speed accuracy.

These criteria are difficult to measure, however the use of a motion capture system to analyse the foot and hexapod base position while it walks would aid in evaluating the effectiveness of the walking algorithm and assist in improving it further.

This same motion capture system can be used to provide more accurate and repeatable testing of the individual foot position and speed to assist in improving the control of each foot.

17.3 Closed Loop Foot Control

Although the hexapod now calculates foot positions in real time, the control of foot position and speed is essentially still open loop as there is no position or speed feedback from the foot tip. An investigation into adding a further control loop to the system to provide position and speed feedback to the controller could improve the response of the hexapod.

One such way of doing this, however not practical in real world use, is to use a motion capture system to analyse the position and speed of each foot tip and send this data back to the control loop.

17.4 Closed Loop Hexapod Control

The true body position and speed of the hexapod is not measured, meaning that the body control is also open. Adding feedback of the body position, velocity and acceleration will further aid in improving the walking algorithms.

This can be done by using the on-board IMU to determine the horizontal walking acceleration, velocity and position through integration of the IMU data. Implementing a Global Positioning System and fusing this data with the IMU data will further add to the accuracy of the hexapod body position control loop.

17.5 Stabilisation Improvement

An improvement of the PID stabilisation may be possible by investigating the performance of adding an angular rate controller to the stabilisation control loop. This may aid in the rejection of the base motion during walking and give the hexapod's body a smoother motion. This would need to be evaluated experimentally to determine whether or not there is an improvement and if the improvement is significant enough to warrant the increased control complexity of the system.

17.6 PID Design

The PID controller was designed using the Ziegler-Nichols tuning method. This is a sufficient starting point, however an improved PID controller can be implemented by taking a more analytical approach and linearising the non-linear hexapod model to obtain the transfer function from the commanded pitch and roll to the actual pitch and roll of the system. This will aid in redesigning the PID control parameters to improve the response of the system.

17.7 Search and Rescue Sensor Payload

Developing a sensor payload will allow the hexapod to be used in a search and rescue environment. Such sensors should include a wireless First-Person-View (FPV) camera at a minimum. The inclusion of an infrared camera will aid in the detection of people in a hazardous environment.

17.8 Independent STM32F4 Remote Control

Controlling the hexapod through a computer can be cumbersome as a laptop is needed at the scene where the hexapod is to be used. It would be beneficial to have an independent remote control.

One idea for such a remote is to use another STM32F4 (or the smaller STM32F0 would be sufficient) with buttons and joysticks connected to it to receive user input such as movement commands and other functions from buttons. This remote could then transmit the commands wirelessly to the hexapod. This does however have the limitation of it being only one way communication, unless a portable LCD screen is fitted to it to provide the user feedback.

17.9 Motor Replacement

The RX28 motors that have not yet been upgraded to the RX64 motors are sufficient for normal use, however for the use of gripping and manipulating objects around the hexapod it would be beneficial to have more torque available. Upgrading the rest of the motors on the hexapod, with the necessary, minor mechanical changes that would be required, will provide the hexapod with the torque it needs.

An alternate, although more expensive, upgrade to the motors is to replace them all with Dynamixel motors from the MX series. These motors allow for the reading of the control table of all motors through one serial command called 'bulk read' [69]. This decreases the time needed to read all motors significantly and would allow for the constant monitoring of motor positions, torques and temperatures (as well as any other motor data). This would allow the dynamic model of the leg to be used in real time to control the motors at precisely the torques required for a desired gripping force.

17.10 Remove 3D Printed Feet

The 3D printed feet are now sufficient for use, however a better and stronger foot would be one made from aluminium. The recommendation is to machine a new link three for each leg. One that ends with a rounded point to act as a foot so that the final leg link and foot are one unit. This will remove the possibility of feet breaking and the hexapod losing functionality.

A Solidworks rendering of the hexapod leg with the 3D printed foot removed and the new link three can be seen in the figure below.

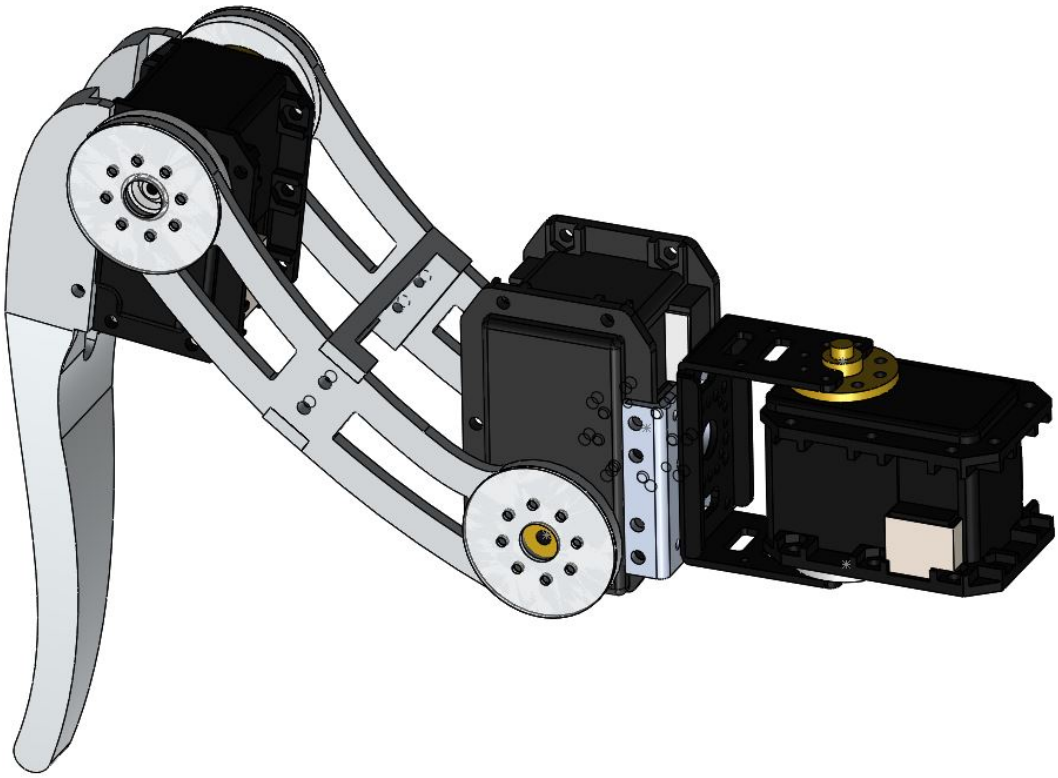


Figure 17.1: Hexapod Leg with 3D Printed Foot Removed

18 References

[1] Wikipedia, *Anthropomorphism*. [Online]. Available: https://en.wikipedia.org/wiki/Anthropomorphism.

[2] A. Ahmed, M. Henrey, P. Bloch, P. Gupta, C. Panaitiu, D. Naaykens, S. Strbac, L. Shannon, and C. Menon, "A miniature legged hexapod robot controlled by a fpga", *International Journal of Mechanical Engineering and Mechatronics*, vol. 1, no. 2, 2013, ISSN: 19292724. DOI: 10.11159/ijmem.2013.001.

[3] M. A. Md. Masum Billah and S. Farhana, "Walking hexapod robot in disaster recovery developing algorithm for terrain negotiation and navigation", *International Journal of Mechanical and Mechatronics Engineering*, vol. 2, no. 6, 2008.

[4] T. Booysen and S. Marais, "The development of a remote controlled, omnidirectional six legged walker with feedback", in *Africon*, ser. Series The Development of a Remote Controlled, omnidirectional six legged walker with feedback, IEEE, 2013.

[5] T. Booysen and F. Reiner, "Gait adaptation of a six legged walker to enable gripping", in *2015 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, ser. Series Gait adaptation of a six legged walker to enable gripping, 2015, pp. 195–200. DOI: 10.1109/RoboMech.2015.7359522.

[6] S. Marais, A. Nel, and P. Robinson, "Reflex assisted walking for a hexapod robot", in *PRASA-RobMech*, ser. Series Reflex Assisted Walking for a Hexapod Robot, Pattern Recognition Association of South Africa, Robotics, and Mechatronics, 2016.

[7] S. Weihe, *Design a high current, lightweight power supply unit with a battery as well as a docking stand with integrated wireless charging for the rarl hexapod robot.* 2019. [Online]. Available: https://en.wikipedia.org/wiki/RS-232.

[8] D. Thilderkvist and S. Svensson, "Motion control of hexapod robot using model-based design", Thesis, 2015.

[9] J. A. T. Machado and M. F. Silva, "An overview of legged robots", Thesis.

[10] M. F. Silva and J. A. Tenreiro Machado, "A historical perspective of legged robots", *Journal of Vibration and Control*, vol. 13, no. 9-10, pp. 1447–1486, 2016, ISSN: 1077-5463 1741-2986. DOI: 10.1177/1077546307078276.

[11] Boston Dynamics, *Atlas robot*. [Online]. Available: https://www.bostondynamics.com/atlas.

[12] ——, *Spot classic robot*. [Online]. Available: https://www.bostondynamics.com/legacy.

[13] ——, *Rhex robot*. [Online]. Available: https://www.bostondynamics.com/rhex.

[14] Generation Robots, *Phantomx ax*. [Online]. Available: https://www.generationrobots.com/en/401679-hexapod-robot-kit-phantomx-ax.html.

[15] A. Krishna, K. Nandanan, S. P. Kumar, S. K.S, and S. P, *Design and fabrication of a hexapod robot*, Conference Paper, 2014.

[16] S. K. K. Chu and G. K. H. Pang, "Comparison between different model of hexapod robot in fault-tolerant gait", *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 32, no. 6, pp. 752–756, 2002, ISSN: 1083-4427. DOI: 10.1109/tsmca.2002.807066.

[17] G. C. Haynes and A. A. Rizzi, *Gaits and gait transitions for legged robots*, Conference Paper, 2006.

[18] S. M. SONG and B. S. CHOI, "The optimally stable ranges of 2n-legged wave gaits", *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS*, vol. 20, no. 4, pp. 888–902, 1990.

[19] X. Ding, Z. Wang, A. Rovetta, and J. Zhu, "Climbing and walking robots", in, B. Miripour, Ed. InTech, 2010, ch. Locomotion analysis of a hexapod robot, pp. 291–310.

[20] J. M. Porta and E. Celaya, "Reactive free-gait generation to follow arbitrary trajectories with a hexapod robot", *Robotics and Autonomous Systems*, vol. 47, no. 4, pp. 187–201, 2004, ISSN: 09218890. DOI: 10.1016/j.robot.2004.04.001.

[21] N. H. DARBHA, "An optimization strategy for hexapod gait transition", Thesis, 2014.

[22] M. M. A. HAJIABADI, "Analytical workspace, kinematics, and foot force based stability of hexapod walking robots", Thesis, 2013.

[23] "Biomimicry, an overview", *The Holistic Approach to Environment*, vol. 5, 2015.

[24] L. E. Parker, "Distributed intelligence: Overview of the field and its application in multi-robot systems", 2007.

[25] *Dynamixel rx-64 manual*, http://emanual.robotis.com/docs/en/dxl/rx/rx-64/, Accessed 22-07-2019.

[26] T. Booysen, *Uct mec4053z digitals notes*, 2016. [Online]. Available: https://vula.uct.ac.za/access/content/group/5504bf4d-2c7a-4c87-9f46-751da4ea8bc8/Notes/MEC4053Z%5C%20Notes%5C%202017.pdf.

[27] A. Mensink, "Characterization and modeling of dynamixel servo", Thesis, 2008.

[28] *Ni labview*, Computer Program, 2015.

[29] FitPC, *Fitpc 2 specifications*. [Online]. Available: http://www.fit-pc.com/web/products/fit-pc2/.

[30] Z. L. Richard M. Murray and S. S. Sastry, “A mathematical introduction to robotic manipulation”, 1994.

[31] K. M. Lynch and F. C. Park, “Modern robotics: Mechanics, planning and control”, 2017.

[32] G. S. Chirikjian, “Rigid-body kinematics”, *Robotics and Automation Handbook*, 2005.

[33] P. Allgeuer and S. Behnke, “Fused angles and the deficiencies of euler angles”, in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ser. Series Fused Angles and the Deficiencies of Euler Angles, 2018, pp. 5109–5116. DOI: 10.1109/iros.2018.8593384.

[34] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, 1st. JOHN WILEY & SONS, INC., 2005.

[35] K. M. Lynch and F. C. Park, “Rigid-body motions”, in *Modern Robotics: Mechanics, Planning and Control*. 2017.

[36] A. N. Norris, “Euler-rodrigues and cayley formulae for rotation of elasticity tensors”, *Mathematics and Mechanics of Solids*, vol. 13, no. 6, pp. 465–498, 2008, ISSN: 1081-2865 1741-3028. DOI: 10.1177/1081286507077982.

[37] M. Jafar, “Generalized screw transformation and its applications in robotics”, *Journal of Advanced Technology Sciences*, vol. 4, no. 2, pp. 81–88, 2015.

[38] S. Sahu, B.B.Biswal, and B. Subudhi, “A novel method for representing robot kinematics using quaternion theory”, in *IEEE Sponsored Conference on Computational Intelligence, Control And Computer Vision In Robotics & Automation*, ser. Series A Novel Method for Representing Robot Kinematics using Quaternion Theory, 2008.

[39] K. A. Loparo and I. S. Vakalis, “D-h convention”, *Robotics and Automation Handbook*, 2005.

[40] M. Serge, T. Patrick, and F. Duquenoy, “Chapter 6 - motion systems: An overview of linear, air bearing, and piezo stages”, in *Three-Dimensional Microfabrication Using Two-photon Polymerization*, ser. Micro and Nano Technologies, T. Baldacchini, Ed., Oxford: William Andrew Publishing, 2016, pp. 148–167, ISBN: 978-0-323-35321-2. DOI: https://doi.org/10.1016/B978-0-323-35321-2.00008-X. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B978032335321200008X.

[41] S. H. Mark W. Spong and M. Vidyasagar, *Robot Dynamics and Control*, 2nd. 2004.

[42] *Solidworks*, Computer Program, 2019.

[43] K. M. Lynch and F. C. Park, “Dynamics of open chains”, in *Modern Robotics: Mechanics, Planning and Control*. 2017.

[44] H. Sadegh Lafmejani and H. Zarabadipour, “Modeling, simulation and position control of 3dof articulated manipulator”, *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, vol. 2, no. 3, 2014, ISSN: 2089-3272 2089-3272. DOI: 10.11591/ijeei.v2i3.119.

[45] *Matlab*, Computer Program, 2019.

[46] C. S. Gurel, “Hexapod modelling, path planning, and control”, Thesis, 2017.

[47] K. Yoneda, “Gait and foot trajectory planning for versatile motions of a six-legged robot”, *Journal of Robotic Systems*, vol. 14, no. 2, pp. 121–133, 1997.

[48] R. L. Williams, “Simplified robotics joint-space trajectory generation with a via point using a single polynomial”, *Journal of Robotics*, vol. 2013, pp. 1–6, 2013, ISSN: 1687-9600 1687-9619. DOI: 10.1155/2013/735958.

[49] *How to select the right two way radio frequency*, https://www.novacommunications.com/2013/11/06/how-to-select-the-right-two-way-radio-frequency, Accessed: 22-07-2019.

[50] yesyes DIY and astro, *Rf1100-232 rf 433mhz transceiver module*, 2019. [Online]. Available: http://www.yesyes.info/index.php/electronics/rf1100-232-rf-433mhz-transceiver-module/.

[51] *2 x wireless rf transceiver module 433mhz cc1101 rf1100*, https://images-na.ssl-images-amazon.com/images/I/61GCLDK7pML._SX679_.jpg, Accessed: 22-07-2019.

[52] Sparkfun and T. Corrine, *Accelerometer basics*, https://learn.sparkfun.com/tutorials/accelerometer-basics, Accessed 29-07-2019.

[53] SENSORWIKI.ORG, *Accelerometer*, https://sensorwiki.org/sensors/accelerometer, Accessed 29-07-2019.

[54] U. of Washington - Software For Embedded Systems, *Mems sensors*, https://courses.cs.washington.edu/courses/cse466/15au/pdfs/lectures/MEMS%20Sensors.pdf, Accessed 29-07-2019.

[55] Wikipedia, *Mems magnetic field sensor*, https://en.wikipedia.org/wiki/MEMS_magnetic_field_sensor, Accessed 29-07-2019.

[56] X. Technologies, *Mti and mtx user manual*, 2008.

[57] InvenSense, *Mpu-9250 product specification*, 2014.

[58] *Mt software suite*, Computer Program, 2019.

[59] K. A. Tehrani and A. Mpanda, “Pid control theory”, in *Introduction to PID Controllers – Theory, Tuning and Application to Frontier Areas*, R. C. Panda, Ed. Central Leather Research Institute, India: InTech, 2012, pp. 213–228, ISBN: 978-953-307-927-1. DOI: 10.5772/2422. [Online]. Available: http://www.intechopen.com/books/introduction-to-pid-controllers-theory-tuning-andapplication-to-frontier-areas/theory-of-pid-and-fractional-order-pid-fopid-controllers.

[60] K. J. Astrom and R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. 2006.

[61] L. Mohammed and M. M. Ahmed, “Spacecraft pitch pid controller tunning using ziegler nichols method”, *IOSR Journal of Electrical and Electronics Engineering*, vol. 9, no. 6, pp. 62–67, 2014, ISSN: 2278-1676.

[62] Wikipedia, *Rs-232*, 2019. [Online]. Available: https://en.wikipedia.org/wiki/RS-232.

[63] J. Gowans, *Stm32f051 dev board*, Department of Electrical Engineering, 2014.

[64] *Easyeda designer*, Computer Program, 2019.

[65] *Hal drivers*, Computer Program. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/user_manual/2f/71/ba/b8/75/54/47/cf/DM00105879.pdf/files/DM00105879.pdf/jcr:content/translations/en.DM00105879.pdf.

[66] L. Technology, *Ltc4013 datasheet*. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/4013fa.pdf.

[67] Microsoft, *Extern (c++)*. [Online]. Available: https://docs.microsoft.com/en-us/cpp/cpp/extern-cpp?view=vs-2019.

[68] *Netbeans*, Computer Program.

[69] Dynamixel, *Dynamixel protocol 1.0*. [Online]. Available: http://emanual.robotis.com/docs/en/dxl/protocol1/#bulk-read.

[70] B. Goodwine, “Inverse kinematics”, *Robotics and Automation Handbook*, 2005.

[71] M. L. Nagurka, “Newton-euler dynamics of robots”, *Robotics and Automation Handbook*, 2005.

[72] M. Zefran and F. Bullo, “Lagrangian dynamics”, *Robotics and Automation Handbook*, 2005.

[73] A. P. Gracia, “Kinematics of robots”, 2007.

[74] D. Klain, “The matrix exponential”, 2017.

A Hexapod Gait Comparison

The gait comparison table is repeated below and an explanation of the decisions behind each value is given.

Table A.1: Hexapod Gait Comparison

Gait	Duty Factor	Speed	Stability	Preferred Surface Type	Interact with Objects	Control Complexity
Free	N/A	5	5	Any	No	5
Tripod	$\frac{1}{2}$	4	2	Smooth	No	0
Wave	$\frac{5}{6}$	0	4	Smooth	No	1
Quadruped	$\frac{2}{3}$	2	3	Smooth	No	2
Gripping	$\frac{3}{4}$	1	1	Smooth	Yes	3

The free gait has been shown previously to mimic the tripod gait on flat ground. However on rough terrain the free gait is capable of optimizing its foot placement and for this reason it is the fastest of all gaits and the most stable. The control required to dynamically determine foot placements based on the environment around the hexapod is complex and requires more sensors to determine the layout of the ground. This is the most complex control of the gaits studied.

The tripod gait has previously been stated to be the fastest gait [20], however it cannot account for rough terrain and as such, overall, it is not as fast as the free gait. It has only 3 legs on the ground at any time so it is statically stable but other gaits with more legs on the ground at the same time are more stable. The tripod gait is simple to control as it is simple the same cycle of motion for each leg offset by a 50% duty cycle.

The wave gait is the slowest gait as it only moves one leg at a time, however this makes it more stable than other gaits (except for the free gait which also only moves one leg at a time but each foot position is optimized).

The quadruped gait is faster than the wave gait as fewer legs are on the ground at the same time, however this also makes it less stable.

The gripping gait is the least stable as the center of mass of the robot is shifted from its physical center.

It may be carrying an object that further shifts its center of mass. It only has three legs on the ground at any one time and it only move one leg at a time making it slow. It it complicated to control as a control algorithm is needed for both walking and for gripping and moving the object it holds.

B Dynamixel Motor Control Table

dynamixel RX-64 ROBOTIS

3-4. Control Table

Control Table consists of data regarding the current status and operation, which exists inside of RX-64.

The user can control RX-64 by changing data of Control Table via Instruction Packet.

	Address (hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM Area	0 (0X00)	Model Number(L)	Lowest byte of model number	R	64 (0X40)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	34 (0X22)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	80 (0X50)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 [0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	240 (0XF0)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36 (0X24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36 (0X24)
RAM Area	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	CW Compliance margin	RW	0 (0X00)
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin	RW	0 (0X00)
	28 (0X1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	CCW Comliance slope	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position	R	-
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered Instruction	Means if Instruction is registered	RW	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)

C GitHub Repository and Video Links

The GitHub repository below has the C++ code from the hexapod in the folder ”HexapodControl”. It has the Java control GUI in the folder ”Hexapod Control GUI”. All MATLAB scripts used to develop and test the kinematics, dynamics and trajectory generation is found in the folder ”MATLAB”. The Solidworks model of the final hexapod is found in the ”Solidworks” folder.

As well as the code, the software to program the RF remote modules [50] and a poster of the hexapod from the 2019 postgraduate research expo is included.

`https://github.com/rchristopher/Hexapod-UCT-Masters`

The documentation for the C++ hexapod code can be found at:

`https://rchristopher.github.io/Hexapod-UCT-Masters`

Videos of the hexapod in operation can be found at:

Hexapod Twisting its Body:	`https://youtu.be/Y7dhTmXj7jc`
Hexapod Height Changing:	`https://youtu.be/q2x5ghlYeB8`
Hexapod on Charging Base:	`https://youtu.be/d-ti27CiAhU`
Hexapod Walking at Speed:	`https://youtu.be/vn0S7XvKXdE`
Hexapod Walking on Slope:	`https://youtu.be/eth4lNTiyJM`
Hexapod Walking with Height Changes:	`https://youtu.be/5solkBVhNds`

D Hexapod Budget

Table D.1: Hexapod Budget

Item	**Quantity**	**Cost**	**Total Cost**
M2x10mm SS cap screws	100	1.67	167.00
Laser Cut Body Plates	1	377.15	377.15
STM32F4-Discovery	3	434.15	1,302.45
PCB Manufacture	1	85.55	85.55
PCB Shipping	0.2	649.00	129.80
Molexs and Inserts	N/A	N/A	241.07
8MHz Crystal	1	6.28	6.28
3.3V Regulator IC	2	2.57	5.14
LTC1480	4	78.08	312.32
MAX232	4	0.00	0.00
FDS4435	2	8.81	17.62
STM32F4 IC	1	0.00	0.00
Surface Mount Components	N/A	N/A	55.24
Solder Paste	1	252.00	252.00
Solder Wire	1	240.00	240.00
		TOTAL	**3 710.82**

Multiple components were sourced from the lab and as such are listed with a cost of R0.00.

The molexs and surface mount components were bundled together as the cost of each component is small.

The shipping cost was slit between others ordering from the same company and as such the shipping cost towards this project is $\frac{1}{5}$ of the total shipping cost because five PCBs were ordered and only one was for this project.

E Project Summary

Walking robots are particularly interesting due to their high degree of flexibility and manoeuvrability. They are able to overcome obstacles which conventional wheeled robots cannot. One such useful application of this is in the field of search and rescue. Sending a robot into a dangerous environment to search for people before sending first responders in can aid in pin pointing the exact location of those in need of help. This decreases the risk to the rescuers involved.

The hexapod robot is a hexagonal, symmetric robot that is made up of six legs with three motors per leg. Each leg is in an anthropomorphic configuration which allows for position control of each foot in three dimensions. At the start of 2018 this movement was achieved by moving each motor to a known position, read in from a text file, at known speeds. This ”brute force” approach to control is problematic as it is very limited in the motion that can be achieved. Only movements that have been pre-defined with movement values saved to a file can be made. Pre-defining every motion that the hexapod can possibly make is time consuming and making slight changes to these motions is difficult. This 2018 hexapod was controlled by a on-board PC running LabVIEW which responded slowly and often crashed, requiring a long amount of time to reset itself. In the case of search and rescue, if the robot fails during use and needs time to restart, it is a waste of crucial, life-saving time.

In order to improve the startup time and reset time of the hexapod the on-board computer was abandoned and an STM32F4 micro-controller was used. This controller starts up instantly and requires less power to run than the PC. The software was coded in embedded C++ on the micro-controller to allow for leg position control and variable walking motion. This began by determining the kinematics of the leg of the hexapod and translating each leg’s kinematics into global, body coordinates through body kinematics. The inverse kinematics was used to control the position of each leg while the inverse differential kinematics was used to control the speed.

A path generation algorithm was developed to produce a smooth walking path for each leg by using a 6th order polynomial function with via points. The start point, mid point and end point of the foot path is defined as well as the walking stride length and step height. Using this information a set of points and speeds is generated for the leg to move through, producing a smooth walking motion.

The ability to control the pose of the hexapod’s body is beneficial for sensors mounted on it. For example, if a camera is mounted to the top of the hexapod, to be able to see where the hexapod is going and to look for people, it is useful to be able to control the angle that the camera points to. By using an inertial

measurement unit (IMU) the pitch and roll of the hexapod body is measured and with a PID controller the foot height of each leg is adjusted to set the pose of the hexapod to the desired angle. Keeping the body of the hexapod level at all times, even if the surface that the hexapod is walking on is not level, is the most common use of this controller and therefore it is known as IMU stabilization.

A remote control system was developed to allow for communication with the hexapod remotely. This was done using a radio frequency transceiver that sent and received serial data between the control computer and the hexapod. A graphic user interface (GUI) was developed to allow for the control of walking among other things.

Printed circuit boards (PCBs) were made for the main controller to interface with the motors, IMU and remote. A battery system was developed for the hexapod by Sven Weihe at UCT which included a wireless charging system. PCBs were made for these components and mounted to the hexapod.

After implementing the different components of the hexapod (motors with walking algorithms, IMU and remote) various tests were performed to validate the performance of the robot.

Leg position control tests were performed to validate the inverse kinematics and motor control algorithms. These tests proved that the foot can be moved to any global coordinate point, with the reach of that leg, accurate to 0.993mm.

Walking tests were then performed to verify the walking path algorithms for distance and speed. This was done by walking the hexapod over a marked out track with markings every one meter and comparing the actual distance the hexapod walked to the expected distance that the hexapod should have walked and comparing the time the hexapod took to walk a known distance and the time it was expected to take. These tests lead to the conclusion that the hexapod walking algorithm was successful and the distance was accurate to ± 130mm. The speed was accurate to 2.43mm/s. Inaccuracies in the walking are accounted for by slipping of the feet on the ground surface which was observed during the tests.

The IMU was tested by applying various disturbances to the hexapod and viewing the response of the pitch and roll on a real time plot. This data showed that with the PID controller the hexapod's pitch and roll stabilized successfully in ± 2.5 seconds after the disturbance with a steady state error of ± 0.0005 rad.

These tests showed conclusively that the hexapod's mechanical and electrical components as well as the mathematical model and software operated as expected. The hexapod is capable of walking over uneven terrain while maintaining a level body. It can be controlled remotely and walk smoothly.

With the recommended addition of sensors such as normal cameras, infrared cameras and a GPS module to the hexapod it will be able to be used, as desired, in a search and rescue environment.

Further recommended improvements include replacing all RX-28 motors with the larger RX-64 motors to increase the torque capabilities of each leg.

F Ethics in Research Form

Application for Approval of Ethics in Research (EiR) Projects
Faculty of Engineering and the Built Environment, University of Cape Town

APPLICATION FORM

Please Note:
Any person planning to undertake research in the Faculty of Engineering and the Built Environment (EBE) at the University of Cape Town is required to complete this form **before** collecting or analysing data. The objective of submitting this application *prior* to embarking on research is to ensure that the highest ethical standards in research, conducted under the auspices of the EBE Faculty, are met. Please ensure that you have read, and understood the **EBE Ethics in Research Handbook** (available from the UCT EBE, Research Ethics website) prior to completing this application form: http://www.ebe.uct.ac.za/ebe/research/ethics1

APPLICANT'S DETAILS		
Name of principal researcher, student or external applicant		Ross Christopher
Department		Mechanical Engineering
Preferred email address of applicant:		CHRROS005@myuct.ac.za
If Student	Your Degree: e.g., MSc, PhD, etc.	MSc
	Credit Value of Research: e.g., 60/120/180/360 etc.	180
	Name of Supervisor (if supervised):	A\Prof Hennie Mouton
If this is a researchcontract, indicate the source of funding/sponsorship		Click here to enter text.
Project Title		Mathematical Modelling, Simulation and Control System Optimization of a Hexapod Robot

I hereby undertake to carry out my research in such a way that:

- there is no apparent legal objection to the nature or the method of research; and
- the research will not compromise staff or students or the other responsibilities of the University;
- the stated objective will be achieved, and the findings will have a high degree of validity;
- limitations and alternative interpretations will be considered;
- the findings could be subject to peer review and publicly available; and
- I will comply with the conventions of copyright and avoid any practice that would constitute plagiarism.

SIGNED BY	Full name	Signature	Date
Principal Researcher/ Student/External applicant	Ross Christopher	signature removed	31/10/18

APPLICATION APPROVED BY	Full name	Signature	Date
Supervisor (where applicable)	A\Prof Hennie Mouton	signature removed	31/10/18
HOD (or delegated nominee) Final authority for all applicants who have answered NO to all questions in Section1; and for all Undergraduateresearch (Including Honours).	T. Bello-Ochende	signature removed	6/11/2018
Chair : Faculty EIR Committee For applicants other than undergraduate students who have answered YES to any of the above questions.			

Page 1 of 2