End-to-end Development of a Robotic Quadruped for Traversing on Challenging Terrains

Final Report

Name: Chui Cheuk Yin
UID: 3035554842
Supervisor: Dr. Pan Jia
Date of submission: 4/18/2022
Abstract

This report entails the end-to-end development of a robotic quadruped that can sense its surrounding terrain and adapt its gait depending on the terrain. Current robotic quadrupeds still struggle to traverse on challenging terrains due to hardware and algorithmic deficiencies, and better algorithms and hardware are needed. For the robotic quadruped to adapt its gait to different types of terrain, sensory feedback of the texture and slope of the terrain needs to work coherently with a decision-making algorithm to enable a robotic quadruped to adapt its gait. Using a novel combination of field-programmable gate array with deep reinforcement learning, a robotic quadruped better suited for traversing on challenging terrains has been designed is in the process of being developed. However, many challenges hindered the progress of developing the entire quadruped which would be addressed within this report. Thus far, the motor controller without the field-programmable gate array has been fully developed and tested; the motor with its stator, rotor, and gearbox has been fully developed; the mechanical design of the quadruped has been completed and its body has been fabricated; a prototype of the quadruped’s leg has been assembled that utilizes belts as a form of transmission. Therefore, assembling the quadruped in its entirety is only a matter of time. It is the hopes that this report will serve as a guide for others in their attempt to build a robotic quadruped (not necessarily for machine learning) in Hong Kong.

Acknowledgements

Thank you, Dr. Pan Jia, for teaching the robotics course which I find to be one of the most interesting courses I have taken, and for supervising my research throughout the year.

I would also like to thank the Department of Computer Science for granting students access to graphics processing units in the GPU farm (Phase 2), without which my experiments would have taken forever to run. I also appreciate the research pathway initiative by the University of Hong Kong and the Department of Computer Science for supporting my research endeavors.

To the technicians at Innowing, thank you for helping me with the fabrications of the motor, especially CS Seto.

Last but not least, I thank Miss Mable Choi for making report writing a less torturous event.

Table of Contents

Abstract .................................................. i

Acknowledgements ......................................... ii

List of Figures ................................................ iii

List of Notations ........................................... iii

1 Introduction ......................................................................................................................... 9

1.1 Legged Robotics in the past ................................................................................................. 9

1.2 Legged Robotics in the present .......................................................................................... 9

1.3 Advancements in Legged Robotics .................................................................................... 9

1.4 Previous Work .................................................................................................................. 10
3.12 Comparison with MIT’s motor for the mini cheetah .................................................. Error! Bookmark not defined.

3.6 Summary ........................................................................................................................ 103

4 Conclusion .......................................................................................................................... 104

References ............................................................................................................................. 104

List of Figures

2.1 Central Pattern Generators ............................................................................................... 8
2.2 Magnetic skin for tactile force sensing .............................................................................. 8
2.3 Permanent Magnet Synchronous motor ............................................................................. 10
2.4 Solenoid magnetic field ....................................................................................................... 11
2.5 Magnetic field generated by opposite coils ....................................................................... 11
2.6 Next opposite coils ............................................................................................................ 12
2.7 Opposite poles attract and like poles repel ...................................................................... 13
2.8 Simple demonstration of motor commutation .................................................................. 13
2.9 Placement of permanent magnets on actual rotor ............................................................ 14
2.10 Actual operation of a PMSM ............................................................................................ 14
2.11 Microcontroller ............................................................................................................... 15
2.12 Hall-effect encoder .......................................................................................................... 16
2.13 Transmission of a sequence of bits .................................................................................. 16
2.14 Clock wire synchronizes data sampling ......................................................................... 17
2.15 SPI protocol .................................................................................................................... 17
2.16 Pads and Traces on a PCB ............................................................................................... 18
2.17 Bending moment diagram ............................................................................................... 19
2.18 Biotensegrity .................................................................................................................. 19
2.19 Normal and shear stress ................................................................................................. 20
2.20 Magnetic skin ................................................................................................................. 20
2.21 Markov Decision Process ............................................................................................... 21
2.22 Neural network ............................................................................................................... 22
2.23 A graphical display of state-space versus cycle ............................................................... 23
2.24 Reward chain .................................................................................................................. 24
2.25 Q-function ....................................................................................................................... 25
2.26 Q-function neural network ............................................................................................. 26
2.27 A figurative description of gradient ascent ...................................................................... 26
2.28 Digital signal processor .................................................................................................. 29
2.29 DSP artificial neural network .......................................................................................... 30
3.30 Body without legs ........................................................................................................61
3.31 Machined components from CNC service ..................................................................63
3.32 Lathe at Innowing ......................................................................................................63
3.33 Reworking components at Innowing ..........................................................................64
3.34 CNC milling machine at Innowing ............................................................................64
3.35 Motor cover being milled ..........................................................................................66
3.36 Machined cover fitted on back housing .....................................................................66
3.37 KHK gears ................................................................................................................67
3.38 Assembled rotor ..........................................................................................................67
3.39 Hand coiled stator ......................................................................................................68
3.40 Winding scheme calculator .......................................................................................69
3.41 Rotor mounted onto back housing ............................................................................69
3.42 Components mounted on reworked front housing ......................................................70
3.43 Four jaw vs three jaw ................................................................................................71
3.44 Assembled motor ........................................................................................................72
3.45 Rotor being reworked ................................................................................................72
3.46 Reworked rotor ..........................................................................................................73
3.47 Fixture for sprockets ...................................................................................................74
3.48 Sprocket being reworked ...........................................................................................74
3.49 Reworked sprockets ...................................................................................................75
3.50 Tapping threads ..........................................................................................................75
3.51 Top and bottom shell ..................................................................................................76
3.52 Shell with leg ................................................................................................................77
3.53 Out-sourced leg component .......................................................................................78
3.54 Sorbothane fixture ......................................................................................................79
3.55 Drilling sorbothane .....................................................................................................79
3.55.1 Bone ........................................................................................................................80
3.55.2 Assembled feet ..........................................................................................................80
3.56 Partially assembled body ............................................................................................81
3.57 First motor rig .............................................................................................................81
3.58 Assembled first motor rig ..........................................................................................82
3.59 Hall-effect encoder .....................................................................................................82
3.60 Current sensor .............................................................................................................83
3.61 Solder paste ................................................................................................................84
3.62 Mosfet ..........................................................................................................................85
List of Notations

\( a_i \)  
\( a_t \)  
\( d_i^t \)  
\( f \)  
\( k_i^t \)  
\( n_t \)  
\( r_i^t \)  
\( w_{ij} \)  
\( x_t \)  
\( y_i^t \)  
\( Q \)  
\( R \)  
\( V \)  

parameter that governs amplitude dynamics of oscillator i  
action at cycle t  
parameter that governs amplitude dynamics of oscillator i at cycle t  
network  
first parameter provided by actor to adjust oscillator i at cycle t  
reward at cycle t  
amplitude of oscillator i at time t  
parameter that governs interaction between oscillator i and j  
state at cycle t  
second parameter provided by actor to adjust oscillator i at cycle t  
Q-function  
reward function  
reward-to-go
$W$ \hspace{1cm} weights of neural network

$\theta^t$ \hspace{1cm} phase of oscillator $i$ at time $t$

$\phi_{ij}$ \hspace{1cm} parameter that governs phase between oscillator $i$ and $j$

$v_i$ \hspace{1cm} oscillator intrinsic frequency

$\rho_i$ \hspace{1cm} oscillator intrinsic amplitude

1 Introduction

1.1 Legged Robotics in the past
Throughout human history, robots that can move with their legs have been a fantasy until recently. Yet, it is rarely about the task of walking but rather, the pursuit of intelligence. Take for example the tin man in Wizard of Oz, or even Frankenstein—society often associates robots that can walk with intelligence because walking is a motion that fundamentally requires intelligence. Unfortunately for them, building robots with legs was beyond their time. Fortunately for us, now is the time.

1.2 Legged Robotics in the present
The 21st century has seen a sudden boom in terms of attention paid to legged robotics, wherein its popularity has been bolstered by the social media presence of quadrupedal robots, robots with four legs, that can do tricks and dance, not unlike a household pet. To the layperson, it might appear that these quadrupedal robots appear out of nowhere, and indeed the commercialization of quadrupeds is a very recent phenomenon. Yet, legged robots existed in the 20th century where their electric origin can be traced back to the Hopper robot developed by Marc Raibert (chairman of Boston Dynamics) in the leg lab at MIT (3D One-Leg Hopper (1983-1984), n.d.). So why is there a sudden incentive for companies to invest in these types of robots? The explosion of interest in legged robotics is due in part to powerful yet affordable electric motors made available by the production of consumer-friendly drones. Today, anyone—with sufficient technical skills and sufficient spare change—could build their own robotic quadrupeds as the availability of powerful electric motors ceased to be a bottleneck towards building one. Along with better algorithms and quicker digital processing hardware, the potential for these robots to improve the human condition is swiftly being realized. Henceforth, it is only natural for research in legged robotics to bloom.

1.3 Advancements in Legged Robotics
But how are advancements in legged robotics made, and what is there to contribute? A common trend one can observe along the timeline of quadrupedal robots can be described as something akin to the butterfly effect: the improvement in unrelated hardware which indirectly improves the functionality of the robot. For example, the creation of better motors allowed better legs control; advancements in computer vision allowed quadrupeds to better gauge the placement of their foot on terrains; or how the introduction of digital control opened the door to bolder and ad-hoc control algorithms. Yet, despite such advancements, traversing on different types of terrain remains a challenging prospect for quadruped robots because at its core, to be able to adapt one's walking behavior on different types of terrain requires more than just programs that instruct how the robot should move its legs. Thus, the problem we seek to solve is not unlike current challenges to achieving general artificial intelligence.
1.4 Previous Work
There have been many robotic quadrupeds in both the industry and academia. MIT is known for being one of the pioneers in using electric motors to pilot their legged robots, the notable examples being the Cheetah 1, Cheetah 2, Cheetah 3, and the mini cheetah (MIT Biomimetics Lab, n.d.). Boston Dynamics have developed the Spot (Boston Dynamics Spot, n.d.) which uses a patented leg design, driven also by electric motors. Another notable electric quadruped is the ANYmal from Anybotics (ANYmal Autonomous Legged Robot, n.d.). Needless to say, these previous quadrupeds serve as inspiration to the quadruped being developed.

1.5 Goals and Objectives
This project aims to continue the trend of advancements in legged robotics via the end-to-end development of a robotic quadruped with new capabilities. End-to-end is defined as the entire fabrication process from the development of the lower-level components, such as the electronics and mechanical housing, to the higher-level components, such as the locomotion algorithm.

1.6 Contribution
The developed quadruped will serve to improve current quadrupeds’ ability to traverse challenging terrains. The improvements will be two-fold: on the hardware side, a novel application of field-programmable gate array to motor controllers with enable robotic quadrupeds to learn and achieve better control of their surroundings. On the software side, a novel control algorithm will be developed that utilizes the field-programmable gate array to achieve learned locomotion on challenging terrains.

1.7 Outline of report
In chapter one, a brief introduction was provided to inform the readers of current trends and developments in the field of four-leg robotics. In addition, an argument was made to convince the readers that the pursuit of robots that walk is as much of a pursue of artificial intelligence as any other field in artificial intelligence. Readers are then directed to related works and an overview of what to expect from this (ambitious) project.

In what is to follow, chapter two will discuss the methodology to achieve this project’s objectives. It will begin by diverging background information on how this methodology came to be, followed by specifics about the methodology.

Chapter three presents progress up to this point. The readers will be taken, end-to-end, through the development process of the quadruped. The chapter will end on plans for the developmental process in the future after the time of writing.

Finally, chapter four concludes the report with a summary of what the report comprises of.

2 Methodology
2.1 Background
Animals can change how they walk on different types of terrain by feeling the ground with their feet and adapting their gait, the cyclic motion of walking, to account for the unevenness of the ground. This phenomenon is a result of the so-called central pattern generators (Figure 2.1)—cells in the spinal cord that generates cyclical movements—which are widely believed to be associated with motor control, the control of legs for locomotion, in animals (Guertin, 2013). Central pattern generators are neural networks that operate by generating a pattern of signals that is directly proportional to the movement of the leg, where the pattern generated by the central...
pattern generator changes as a function of what the organism senses. So, how the ground feels to the animal might influence their gait by changing the pattern generated by their central pattern generators. These patterns are learned by the animal subconsciously via trial and error by associating what they feel with their progress on the terrain. And central pattern generators allow for learning because they are neural networks. With sufficient sensors, perhaps a magnetic skin, and hardware that can emulate neural networks, one could emulate this way of walking for robotic quadrupeds.

Recently, a lab at HKU developed a magnetic skin for tactile force sensing (Yan, et al., 2021); the magnetic skin, shown in Figure 2.2, senses forces applied onto the skin from different directions using magnetic sensors. Suppose one applies the magnetic skin onto the sole of a robotic quadruped’s foot, then one could potentially use the sensor’s feedback to emulate how animals adapt their gaits on different types of terrain. However, these sensors are nonlinear hence they are not readily integrable with existing lower-level controllers such as PID (proportional, integrative, derivative control). The challenge, then, becomes how to integrate these sensors with said lower-level controllers, which in turn controls the motor that actuates a robotic quadruped.

Machine learning is a field in computer science that has taken precedence in the 21st century due to a vast number of incredible feats that impressed society at large. However, despite the popularity of machine learning, most robotic quadrupeds today operate via a mixture of control theory, optimal control, and optimization—classical methods for controlling robots that, when applicable, are guaranteed to work, yet are limited in their applicability. In contrast, machine-learned controllers are less limited but offer fewer guarantees when used to control a robot.
One of the more popular methods in machine learning for robotics is reinforcement learning which, as the name suggests, allows a robot to learn via iterative trials of the same task. Like other machine-learned controllers, controllers generated with reinforcement learning are solemn if ever used to control a real robotic quadruped. The problem with reinforcement learned controllers is their inability to perform satisfactorily when tasked with operating on environments the robot has not been trained on, because in reinforcement learning, one trains the robot on solving one and only one task. Consequently, if the task changes a little, perhaps the friction of objects changed a little, then the algorithm might fail to solve the task. This is known as the problem of generalization in reinforcement learning because the controller fails to generalize to tasks similar to the task the controller has been trained on. In summary, something extra that could help generalization is needed if a machine-learned controller is to work.

The quadrupeds we will consider are those electronically controlled. Electricity is a result of current traveling through wires as a result of a potential difference between two ends of the wire. The potential difference, also known as voltage, is usually measured relative to a ground—a part of the circuit with zero potential. When a current travels through a wire, a magnetic field surrounding the wire is generated. This will prove to be useful when motors for the quadruped are considered.

2.2 Introduction
This project involves the end-to-end development of a robotic quadruped; hence, the methodology will cover a range of topics from electrical engineering, mechanical engineering, and computer science. Generally, hardware is either controlled or to query data from. For hardware that requires control, details will be provided on the workings of the hardware and how control is achieved. As for hardware that provides data, details of how the data is queried will be provided.

2.3 Motor design and control
Legged robots are different from other robots in that the actuator, the electronic that generates force, has to support the robot’s own weight in addition to doing useful work, such as exerting energy to push a box. A motor is a good candidate for such an actuator because, firstly, it is economical, and secondly, compared to other actuators it is easy to control for the amount of power it provides. Unfortunately, most motors that are affordable do not meet the minimum force requirement that robotic quadrupeds require. Therefore, a custom motor needs to be developed.

2.3.1 Permanent Magnet Synchronous Motor
The type of motor to be developed is a permanent magnet synchronous motor (PMSM), shown in Figure 2.3.
In a PMSM, the stator oversees the rotation of the rotor, a component meant for rotating that is connected to the stator through a shaft in the center. The stator is composed of coils of brown copper wires of which generates a magnetic field when a current flows through the wires, a phenomenon known as induction (Figure 2.4).

*Figure 2.3 Permanent magnet synchronous motor*
Initially, no current flows through the wires hence the stator does not have a magnetic field. But, when current flows through the coils on opposite ends, a magnetic field along the axis that intersects the coils is generated (Figure 2.5).

If the same process is applied to the opposite coil pair next to the current opposite coil pair, the resultant magnetic field could be shifted (Figure 2.6).
Figure 2.6 Top left: Current flows through the next opposite coil pair. Top right: Green arrow represents the resultant magnetic field due to the new opposite coil pair. Cancel the north and south poles in the middle. Bottom: The resultant magnetic field after summing up the two green arrows. Observe that it has a different angle than the original green arrow.

By repeating this process of activating new opposite coil pairs and, in other cases, deactivating opposite coil pairs, the resultant magnetic field—represented by the green arrow—of the stator can rotate a full three hundred and sixty degrees around the center of the stator. This will form the basis of a process known as motor commutation. Motor commutation uses a principle where two magnetic fields, when oriented at an angle from one another, create a force of attraction and repulsion depending on the orientation of the poles (Figure 2.7).
Opposite poles attract and like poles repel. Green arrow represents the resultant magnetic field of the stator. Purple arrows represent the resultant field of the rotor generated by two permanent magnets represented by the red and blue block. Convince yourself that if the stator was fixed in place, then the rotor would rotate such that the two resultant vector aligns and points in opposite directions.

When this principle is applied to the PMSM, observe that the rotor would move such that its magnetic field is aligned with the stator’s magnetic field if the stator was held in place. Henceforth, to control a PMSM, the control algorithm must adjust the magnetic field of the stator such that it is always leading the magnetic field of the rotor as follows:

Simple demonstration of motor commutation. Green as resultant magnetic field of stator, purple is that of the rotor. Note that the green vector is being controlled, whereas the purple vector is reacting to the orientation of the green vector.

For the example shown in Figure 2.7, the rotor contains only a pair of permanent magnets on opposite ends. In reality, these permanent magnet pairs are placed all around the rotor (Figure 2.9) and complicate control. The
algorithm that allows for the application of motor commutation to more complicated rotors is called field-oriented control.

![Figure 2.9 Placement of permanent magnets on an actual rotor. Treat it as a 2D image.](image)

2.3.2 Field-oriented Control
As its name suggests, field-oriented control (FOC) seeks to orient the magnetic fields of the stator such as to generate motion in the rotor. In actual PMSMs, their stator windings can be divided into three phases, where each phase consists of a single wire.

![Figure 2.10 Actual operation of a PMSM. Each phase of windings is represented by three different colors and symbols: Phase A is red, phase B is blue, and phase C is green. In this instance, only phase A and B have current flowing through them. North and](image)
The current for each phase enters via their “Start” and exits via their “End” label respectively. For example, current for phase A enters via the “Start A” label and exits via the “End A” label. If the current was to flow through phase A and phase B, the coils would generate the magnetic poles as shown. These poles would interact with the permanent magnets on the rotor, where the yellow and purple arrows are the manifestations of these interactions. Thus, in this instance where current flows in phase A and phase B, the rotor would rotate counterclockwise. Different variations of magnetic poles on the stator can be generated by changing how the current flows into each phase. Therefore, to control the rotation of the rotor in a PMSM, field-oriented control adjusts how the current flows into the stator such that the magnetic poles generated by the stator cause continual rotation of the rotor—this can be done by a microcontroller. However, there remains the problem of knowing the position of the rotor, because the stator magnetic poles configuration as shown in Figure 2.10 is only applicable to the current orientation of the rotor; if the rotor was at a different orientation, the same stator magnetic poles configuration might not work. All in all, a device, known as an encoder, that could inform a controller of the position of the rotor is needed, as well as a microcontroller to allow for programmatic control of the current that flows through the stator.

2.3.3 Microcontroller
One can think of a microcontroller as a small computer in the form of a chip that fits onto the surface of a thumb.

![Microcontroller on a printed circuit board. Figure taken from (Gudino, 2018).](image)

Programs are uploaded onto the microcontroller of which is attached to a printed circuit board, basically a circuit on a small silicon board. The program instructs the microcontroller to interact with other electronics attached to the printed circuit board, allowing the programmer to control electronic devices via their program. Likewise, a microcontroller can be used to allow a programmer to control the currents through the stator via code.

2.3.4 Encoder
An encoder is an electronic device that tracks the angular position of the rotor and returns the angular position as information. Ideally, the encoder avoids contact with the rotor otherwise continual operation of the motor might damage the encoder in the long run; one such encoder is the hall-effect encoder. A hall-effect encoder senses the position of the rotor by first attaching a magnet to the rotor and then senses the magnetic field of the attached magnet with an integrated circuit, basically a circuit that is small.
These integrated circuits usually communicate via the Serial Peripheral Interface (SPI), a communication protocol that allows for reliable data transfer between two devices—a master device and a slave device, so-called because the master device is in control of the slave device like how our controller is the master device and the encoder the slave device. When it comes to electronics, communication is achieved by transmitting bits through a wire. A bit is a number that can take on two values, zero or one, and we can assign arbitrary meaning to these bits to convey information. For instance, in the case of a device that monitors the charge capacity of a battery, zero could indicate “out of battery” and one could indicate “battery full”. Hence, the battery monitor needs only to send the corresponding bits to inform other devices about the current state of the battery. Obviously, we could increase the amount of information sent by using a sequence of bits instead of merely one bit.

For digital devices such as our microcontroller and encoder, a wire connected between them is used to transmit bits. When current flows through the wire, the potential difference between it and ground is non-zero—this indicates a bit of value one. If no current flows through the wire, then this indicates a bit of value zero. But, herein lies a problem: when the devices are not communicating, no current would flow through the wire, yet no current is also an indication of a bit of value zero. SPI solves this conflict by introducing another wire, the serial clock (SCLK) wire.
The clock wire is connected between the two devices and synchronizes the reading and writing of bits between the master and slave. It works by generating pulses—a signal that goes from zero to a non-zero potential, then back to zero—in a periodic fashion as demonstrated in the graph labeled “CLOCK”. The moment where the pulse goes from zero to a non-zero potential is called the rising edge, and it is at this moment that either the master or slave reads the bit sent from the other corresponding device. This sampling operation is shown as green arrows in Figure 2.14. So, whenever a rising edge is detected, both devices know that a bit is being sent and that the communication is intended, henceforth the ambiguity of whether the bit is intended to be zero or communication not intended is resolved.

Because SPI is a full-duplex protocol, the master and slave need to be able to send data to one another at the same time. This can be done by simply using two wires to transmit information: one wire is used by the master to transmit bits to the slave, and another wire is used by the slave to transmit bits to the master. To summarize, we have the serial clock wire that synchronizes communication between the master and slave devices, and two wires for communication between the two devices. Additionally, most SPI contains another wire, the chip select wire, which is irrelevant to this project. The below figure provides a visual summary:

![Diagram showing SPI protocol connections]

**Figure 2.15 SPI protocol. CS stands for chip select. SCLK stands for serial clock. MOSI stands for master-out-slave-in, referring to the fact that the master device uses this wire to transmit bits out of the master device and into the slave device. Similarly, MISO stands for master-in-slave-out, wherein the slave uses this wire to transmit bits out of the slave device and into the master device. Figure taken from (Grusin, n.d.).**

### 2.3.5 Printed Circuit Board

We have determined that the control of a PMSM requires controlling how current flows through the stator. To control the current, a microcontroller is used and an encoder provides feedback as to the position of the rotor relative to the stator. To conclude the methodology on motor design, this sub-chapter will consider how the
electronics, such as the microcontroller and encoder, are connected. It was mentioned in chapter 2.3.3 that a microcontroller is placed onto a printed circuit board to communicate with other electronics. The process of communication between electronics is clarified in chapter 2.2.4 to be the transmission of bits and perhaps other signals through wires connected between devices. However, wires are large relative to devices such as microcontrollers and encoders, and this becomes problematic when signals sent through the wires are high in frequency. The problem is analogous to a light ray entering a long bending tube: light will reflect and refract within the tube, and if the tube is long enough, no light ray will succeed in escaping the tube. In this analogy, the bit and wire are analogous to the light ray and the tube respectively.

To circumvent this problem, one could mount these components onto a circuit board and use traces instead of wires. Traces, as shown in the figure below, are coppers laid flat on a silicon board and act as wires between two electronic devices. Pads are used to mount the electronic devices to the board; hence traces are used to connect pads electronically, which in turn connects the electronics. Traces are preferred over wires in a small device, high-frequency scenarios because traces are smaller than wires, and as a result, suffer less from signal loss than wires.

![Figure 2.16 Pads and traces on a printed circuit board.](image)

2.4 Leg design
Following the motor’s development is the design of the legs for the quadruped. Since the algorithm for leg control is based on central pattern generators, it is only fitting that the design of the legs is, likewise, biologically inspired.

2.4.1 Biotensegrity
Biotensegrity is the study of how an organism’s mechanical properties allow them to function. From the standpoint of evolution, the design of animals is a result of a long-running optimization program—survival in a competitive world—which optimizes for increased load carrying and reduced metabolic costs. In other words, evolution finds the least energy-requiring solution to tasks and will become the norm once found.

Most engineering disciplines use Newtonian mechanics, so mechanics based on the application of Newton's Laws of motion, as the basis for their understanding and calculations. Indeed, Newtonian rules serve civil or mechanical engineers well when they only need to consider the structural integrity of buildings and beams. However, the structure of most biological organisms does not abide by Newtonian mechanics and are, thankfully, non-Newtonian in nature; otherwise, we would find ourselves crushing under our weight! Musculoskeletal systems, systems that combine muscles and bones, can achieve structural robustness greater than engineering materials,
most of which by themselves are stronger than bones. It is for this reason that biotensegrity is the design methodology adopted in the MIT Cheetah 3 (Bledt, et al., 2018).

The designers of MIT Cheetah 3 hypothesized that the synergistic co-location of bones and tendons, rope-like tissues that connect muscles to bones, helps to reduce the bending moment at the bone structure.

![Diagram showing bending moment](image1)

*Figure 2.17 Diagram that shows bending moment. Top: A beam without any force applied. Bottom: A beam that is bending under force applied topside. The moment refers to the rotation of the beam about the left and right ends of the beam. A bone is similar to a beam. Figure taken from (Bending Moment, 2020).*

Clearly, a beam is better suited for sustaining force applied along its axis, the line parallel to the length of the beam, as opposed to sustaining force applied perpendicular to its axis. Applying this principle to leg design, consider the following figure:

![Diagram of tendons on human's foot](image2)

*Figure 2.18 Tendons on human's feet. Figure from (Sangbae Kim, 2014).*
Figure 2.18a shows a human’s feet with bones and tendons attached. The purpose of this figure is to show that without those tendons, there will be bending moments within the foot. Figure 2.18c shows a bending moment diagram without tendons attached. The important thing to note is that the black arrows, representative of forces within the bones, point in opposite directions, with different magnitudes, and do not intersect with one another. This means that the resultant force is a rotational one—a bending moment. Figure 2.18d shows that same case except with tendons attached to the feet. In this case, observe that the black arrows within the bone, represented by the grey bar, points in one direction, and on the other hand, the tendon, represented by the red bar, contains black arrows pointed in the opposite direction. By comparing Figure 2.18c and 2.18d, the inclusion of tendons has effectively transmitted the forces in the bones that would have generated moment onto the tendons. Consequently, no moments are induced in the bones. Similar ideas can be used to design the legs for a robotic quadruped.

2.4.2 Magnetic Skin Sensor

It was mentioned in the introduction of this chapter that a method for sensing the ground, its texture and slope, is required for adaptive locomotion. Furthermore, a magnetic skin sensor has been developed at a lab at the University of Hong Kong that could detect sheer and normal force upon contact.

![Difference between Normal Stress & Shear Stress](image)

*Figure 2.19 Difference between normal and shear stress. Note the difference between stress and force. The large blue arrow that points down on the left-hand side is a normal force because it is applied in a direction normal to the surface. This normal force induced a normal stress in the material. Likewise, the large blue arrow that points to the right on the right-hand side is a shear force because it is applied in a direction parallel to the surface. This shear force induced a shear stress in the material. Force is an extrinsic phenomenon whereas stress is an internal phenomenon experienced by the material as a result of external force. Figure from (Engineering, 2019).*

Although sensors that imitate skin have been around for quite some time, they either detect one of normal force or shear force or a combination of the two as one output. The newly developed magnetic skins are different in that in addition to detecting both normal and shear force, they could separate normal from the shear force.

![Demonstration of the magnetic skin](image)

*Figure 2.20 Demonstration of the magnetic skin. The blue layer is a magnetic film that has a magnetic field and is flexible. Grey area refers to a medium between the magnetic film (blue) and a hall sensor (orange). The virtual magnetic needle refers to what the hall sensor senses at that point, note that the hall sensor does not have a magnetic field. Figure from (Yan, et al., 2021).*
Referring to Figure 2.20, when the sensor is at its initial state, the magnetic fields are as shown. In the middle figure, when an object applies a force $F_z$, normal to the magnetic film, the virtual magnetic needle is oriented at an angle $\alpha$ to the original virtual magnetic needle. This angle is proportional to the normal force/stress. In the rightmost figure, when a force $F_x$ is applied to an object, a shear force is applied onto the magnetic film via the friction between the object and the magnetic film. This shear force is proportional to the x-axis component of the vector $d$ shown in red in the figure. With proper calibration, this sensor can provide accurate normal and shear force feedback upon contact of the quadruped’s feet with the ground.

However, magnetic skin sensors are not readily integrable with lower-level controllers such as a PID controller. This is because PID is a linear controller and magnetic skin sensors are nonlinear. One could use the magnetic skin sensors with a higher-level controller, such as an algorithm that runs on the main computer, but the bandwidth associated with the higher-level controller would be too low to achieve the closed-loop control required to fully utilize the skin sensor. For example, when humans touch an object to manipulate it within their grasp, they are not actively thinking about how the object is gripped—otherwise, even the simple activity of holding a cup might take a long time! Instead, we “intuitively” know how a cup is gripped through practice. One can think of the “intuition” as a lower-level controller, and active thinking as the higher-level controller. Therefore, in order to be able to use skin sensors for locomotion, the skin sensor must be able to interface with a lower-level controller. Furthermore, much like how one learns to hold a cup, an algorithm that can allow the robotic quadruped to learn to use skin sensors to achieve locomotion must be present.

2.5 Reinforcement Learning

Prior to this sub-chapter, we considered the motor, leg, and sensor design. We further discussed the importance of an algorithm that can enable a quadruped to teach itself how to use skin sensors for locomotion, but how does a robotic quadruped teach itself? The purpose of this sub-chapter is to answer this question beginning with an introduction to reinforcement learning.

2.5.1 Introduction to reinforcement learning

Reinforcement learning is the process of learning via trial and error, where a form of reward is to represent how well a task is accomplished. Reinforcement learning originates from the field of optimal control, which as the name suggests, studies how a control task can be done optimally. To define optimality, a task is deconstructed into a reward function which is proportional to how well the task is done. For example, if the task is to teach a dog how to sit, one would give the dog a high reward when it is sitting, and no reward when it is not sitting. However, this example ignores the larger picture, which is how do we get the dog to sit in the first place?

If one were to teach a dog how to sit from scratch, one could wait for the dog to sit by itself and reward the dog at those rare moments in time when the dog decides by itself to sit. A smarter and less luck-reliant approach would be to use rewards to guide the dog to sit. If we decompose the motion of sitting into a chain of movements, one could create a routine where, if followed, leads the dog to sit. Then, one can simply guide the dog through the routine with small rewards along the way, and give the dog a gigantic reward when it eventually sits. In this method for training a dog to sit, we first generated a routine for the dog to follow. We can generalize this method to other scenarios via the Markov Decision Process framework.
In this decision-making framework, an agent, so something that has agency, can decide to take any action at a state. States and actions are representative of the problem and change depending on the problem. The environment is what the agent influences with their actions, and upon taking the action at a state, a new state will be reached, and the agent will receive a reward. In our previous example, the dog is the agent, actions correspond to the dog’s control over its own body, the environment would be the dog’s body and the floor, the state would be the pose of the dog, and the reward is given by a human as a response to the dog’s actions at a given state. This process continues and can be summarized as follows:

1. Take an action at a state.
2. Receive a reward for taking the action at that state.
3. Arrive at a new state as a consequence of the action.
4. Repeat.

Note that while this cycle from step one to step four continues, at each step of the cycle a reward is received. The goal of reinforcement learning is to train an agent that can maximize the cumulative sum of rewards till the cycle ends. The ending of the cycle usually corresponds to reaching some desired states, such as when the dog finally sits. Hence once a goal is reached, the cycle need not continue.

2.5.2 Applied reinforcement learning

When it comes to robotics, the agent is usually a function that takes as input the current state and outputs an action. A neural network is a function approximator that maps a vector of numbers into another vector of numbers and is commonly shown as connections between nodes as seen on the left-hand figure of Figure 2.22.
A neural network is just a sequence of matrix multiplication. In this simple neural network, only one matrix multiplication is involved. To be precise, each forward pass of the neural network involves a matrix multiplication between the input, a vector of numbers \( \mathbf{x} = [x_1, x_2, x_3, x_4] \), and a matrix \( \mathbf{W} = [w_1, w_2, w_3, w_4] \) known as the weights of the neural network, represented by lines in the figure. These weights are numbers that participate in the matrix multiplication and can be adjusted such that the output of its matrix multiplication with the input produces a desirable output \( [a_1, a_2, a_3] \). Conventional neural networks also include the addition of biases (represented by the \( b \)) which is there to offset the outputs of the matrix multiplication. More complicated neural networks involve activations, which is simply applying a function to the output layer. Figure taken from (Jordan, 2017).

In this neural network parameterized by weights \( \mathbf{W} \), there is an input layer that corresponds to our states. These states are represented by numbers in the real domain, \( \mathbf{x} = [x_1, x_2, x_3, x_4] \), which are then multiplied by the weights \( \mathbf{W} \) between the input and output layer to form outputs in the output layer \( \mathbf{a} = [a_1, a_2, a_3] \). This process can be described mathematically as \( \mathbf{a} = \mathbf{f}_W(\mathbf{x}) \) where \( \mathbf{f} \) denotes the series of matrix multiplication between the input and the neural network parametrized by \( \mathbf{W} \). If we consider the outputs \( \mathbf{a} \) as our agent’s actions, then the neural network is a function \( \mathbf{f} \) that maps from state \( \mathbf{x} \) to an action \( \mathbf{a} \). In other words, the neural network \( \mathbf{f} \) parametrized by weights \( \mathbf{W} \) is our agent.
Recall that in a Markov Decision Process (MDP), an agent takes actions in an environment to receive rewards till the cycle ends. Let $t$ denote the number of cycles that have passed. If one were to use our neural network agent, parametrized by $W$, in an MDP, then, at cycle $t = 0$ the agent would be in state $x_0$. The agent will proceed to take an action $a_0 = f_W(x_0)$, receive reward $r_0 = R(x_0, a_0)$ which is a function of the state-action taken, and arrive at a new state $x_1$ as a result of taking action $a_0$ in state $x_0$. From here onwards, cycle one begins, and the above procedure is repeated for $t = 1$, $t = 2$, $t = 3$, ..., all the way until either the agent has reached the goal or other termination conditions, such as time constraints, are met. Figure 2.23 shows graphically how a trajectory is generated as the agent continues to act in the MDP. By following the trajectory, a chain of rewards $[r_0, r_1, r_2, r_3, ...]$ is generated and one can sum them up to calculate the accumulated reward earned by the agent—this is the so-called reward-to-go $V(x, W) = \sum_t r_t$ where $r_t$ is the reward the agent received when taking an action $a_t$ at cycle $t$. This process is shown graphically in Figure 2.24. Intuitively, $V(x, W)$ represents the cumulative reward our neural network agent would receive from state $x$ onwards. The challenge, then, is how can one adjust the weights $W$ parametrizing the agent’s neural network so that its reward-to-go $V(x, W)$ increases.
To solve this challenge, reinforcement learning practitioners learn another variant of the reward-to-go. Instead of using $V(x, W)$ directly, they use the Q-function $Q(x_t, a, W) = R(x_t, a) + V(x_{t+1}, W)$ where $x_{t+1}$ is the state that one arrives at when taking action $a$ at state $x_t$. Note that in prior paragraphs, the action has always been generated by the agent. Yet, in this equation, the action $a$ can be any action. By comparing the Q-function $Q(x_t, a, W)$ and reward-to-go $V(x_t, W)$ for the same state $x_t$, observe that the Q-function is simply the reward $R(x_t, a)$ due to taking action $a$ in state $x_t$ added to the reward-to-go for the next state $V(x_{t+1}, W)$. This allows us to calculate the value of the Q-function by first taking a random action $a$ in the MDP, transition to the next state $x_{t+1}$, then letting the agent take actions from $x_{t+1}$ onwards as if the state encountered by the agent when it first entered the MDP is state $x_{t+1}$. Figure 2.25a shows this calculation in terms of the state-space.
Figure 2.25 Part a show the calculation of the Q-function in terms of the state-space. Part b shows how by taking different random actions in the same state $x_0$, new trajectories with their own Q-values are formed.

Figure 2.25b shows how the introduction of the Q-function has enabled the possibility of using random actions to better understand the MDP. Each random action will lead to a different Q-value (output of the Q-function) because not only is the reward $R(x_t, a)$ due to taking random action different for every action $a$, but also because the state transition from $x_t$ to $x_{t+1}$ after taking action $a$ in state $x_t$ is likewise different for every action. By trying out random actions at the same state, for example, the state $x_0$ shown in Figure 2.25b, the Q-values for different actions are found. This will be useful because it is only by comparing the Q-values for different actions that one can know which action is better, the best of which the agent will be trained to use. Hence, one needs to explore the MDP by taking several random actions to generate a set of Q-values (Figure 2.26a).
Figure 2.26 Part a shows the different trajectories due to taking random actions at the first state. Part b shows the steps for training a neural network to emulate the Q-function. Part c shows figuratively how a neural network might interpolate between Q-values.

Suppose a neural network, different from the agent’s neural network, is used to approximate \( Q(x_t, a, W) \), a process described in Figure 2.26b, the neural network approximation of Q can then serve as an interpolation between actions that are in-between the convex hull defined by trajectories used to train the neural network (Figure 2.26c). Interpolation is the Q-value generated by the neural network approximation of Q when an input given to the neural network is not an input that it has been trained on.

Figure 2.27 A figurative description of gradient ascent. Denote the agent’s action in state \( x_0 \) as \( a \). Left hand side shows how the neural network approximation allows for interpolation the Q-value of \( a \) between the Q-values of the random actions taken. Right-hand side shows how gradient ascent can be used to adjust the agent’s action \( a \) such that the adjusted action will have a
Recall that the goal is to adjust the weights of the agent’s neural network such that its reward-to-go, or equivalently its Q-value, increases (this equivalence is true because the Q-value is just a reward added to the reward-to-go). With a neural network approximation of the Q-function, one can optimize the agent’s neural network by performing gradient ascent on the neural network approximating \( Q(x_t, a, W) \). To begin, it is known from elementary calculus that the positive gradient of \( Q(x_t, a, W) \) with respect to \( a \) points to a direction that, if one moves \( a \) along, increases the value of \( Q(x_t, a, W) \), a process described figuratively in Figure 2.27. The gradient \( \frac{dQ(x_t, a, W)}{da} \) exists because the neural network approximating \( Q(x_t, a, W) \) just consists of a chain of matrix multiplications, additions, and activation functions (provided the activation function has a gradient). From the gradient \( \frac{dQ(x_t, a, W)}{da} \), one can, using the chain rule, derive the gradient of \( Q(x_t, a, W) \) with respect to \( W \); similar to \( a \), if one moves \( W \) along the direction of the gradient, the value of \( Q(x_t, a, W) \) will increase and the goal “adjust the weights of the agent’s neural network such that its reward-to-go, or equivalently its Q-value, increases” is achieved. Below is the application of the chain rule:

\[
\frac{dQ(x_t, a, W)}{dW} = \frac{dQ(x_t, a, W)}{da} \cdot \frac{da}{dW}
\]

The gradient \( \frac{da}{dW} \) exists because the agent’s neural network parametrized by \( W \) is used to generate \( a \) and it has already been determined that the gradient from a neural network exists. Likewise, because the gradient \( \frac{dQ(x_t, a, W)}{da} \) exists due to the Q-function being approximated by a neural network, the gradient \( \frac{dQ(x_t, a, W)}{da} \) as a result of the multiplication between two existing gradients also exists. Software such as PyTorch and Tensorflow can automatically compute these derivatives to update the neural network agent’s weights \( W \) such that after updating \( W \) using this derivative, the action \( a_2 = f_{W_{post-update}}(x_t) \) taken by the agent after the update will have a greater Q-value than the action \( a_1 = f_{W_{pre-update}}(x_t) \) taken prior to the update, such that \( Q(x_t, a_1, W_{pre-update}) \geq Q(x_t, a_1, W_{pre-update}) \). This is the basis of a process called Q-learning.

2.6 Field-programmable gate array

In the chapter prior to reinforcement learning, the topic of magnetic skin sensor was introduced, and it was concluded that two things needed to be addressed before skin sensor could be utilized for the control of robotic quadruped: 1. The magnetic skin sensor must be able to interface with a lower-level controller, and 2. The robotic quadruped must be able to teach itself to utilize magnetic skin sensors for traversing challenging terrains. The sub-chapter on reinforcement learning addressed the issue of learning, and this chapter will address the issue of interfacing with a lower-level controller. This feat is achieved with a field-programmable gate array.

2.6.1 What a field-programmable gate array is and why it matters

Field-programmable gate array (FPGA) is a circuit within a chip that is programmable. Essentially, a programmer can write specify a set of instructions, in code, which describes how they want a circuit to behave. Then, the set of instructions is compiled into a configuration file for the FPGA that is uploaded to the FPGA via a serial communication protocol. On startup, the FPGA will configure itself according to the configuration file and become the circuit that it has been configured to become.

In addition to being what amounts to programmable hardware, another merit of field-programmable arrays is their hardware density. Xilinx, a leading manufacturer and the inventor of FPGA, has the technology to fill their FPGA chips with a lot of digital signal processors. These digital signal processors can also be configured by the
configuration file to connect to one another, or to connect to other hardware within the chip. A block diagram of a digital signal processor is shown in Figure 2.28.

![Digital Signal Processor Diagram](image)

Figure 2.28 Digital signal processor. Labels are self-explanatory. Figure from (Xilinx).

A digital signal processor is necessary because operations like multiplication is not easily achievable in software. For example, when doing multiplication on a computer, doing the multiplication purely with logic in C++ will take a long time to compute compared to hardware built specifically for multiplication. Therefore, hardware such as digital signal processor are invented to accelerate computationally time. The fact that FPGAs contain digital signal processors (DSPs) will prove useful for applying magnetic skin sensors to lower-level control.

### 2.6.2 Nonlinear lower-level control with FPGAs

An artificial neural network is a nonlinear controller that can be trained to control any system, even as a lower-level controller. Because neural networks are nonlinear, it can take magnetic skin sensors as input to generate a controller output (provided a means exist for training the neural network). However, traditionally artificial neural networks are not used for lower-level control because the tread today revolves around using graphics processing units (GPUs) to run neural networks. GPUs are not ideal for lower-level control because they have latency revolved around memory read and write overheads, as they were built primarily for graphics and rendering. A high latency is bad for lower-level control because lower-level control revolves around closed feedback loops, so they need to iterate quickly to better control a system. Moreover, GPUs are big and are not meant for soldering on a printed circuit board such as that of a motor controller.

Instead of using GPUs, one can use FPGAs for running neural networks. FPGAs are chips hence they can be added to the printed circuit board of a motor controller. More importantly, they are better suited for running neural networks than GPUs. In the remainder of this sub-chapter, how a FPGA can run a neural network and the advantages of doing so will be discussed.

Firstly, an FPGA can run an artificial neural network because it can become the artificial neural network. As mentioned in the chapter on neural network, a neuron in a neural network consists of a multiplication operation followed by an addition operation. Both operations are contained within one digital signal processor as shown by
the existence of a multiplier and adder in Figure 2.28. Therefore, one can think of a DSP as a neuron in a neural network. Because an FPGA contains many DSPs, each of which can be run in parallel, one can use an FPGA to evaluate a layer of a neural network in parallel because a layer of an artificial neural network consists of many neurons, and each neuron is equivalent to a DSP. Figure 2.29 shows a neural network where each neuron is replaced by a DSP to illustrate the idea of turning an FPGA into a neural network.

![Figure 2.29 An artificial neural network that has its neuron replaced by digital signal processors.](image)

In the neural network emulated by an FPGA, as long as there are enough DSPs to match the number of neurons in a layer, each layer of the neural network can be evaluated in parallel. Because each DSP has low latency, an entire layer of a neural network can be computed with low latency. And if the layers in the neural network are kept low, the entire neural network can be computed with low latency.

Thus, by converting an FPGA into a neural network and soldering the FPGA onto the motor controller printed circuit board, one can essentially add an artificial neural network that has low latency onto the motor controller. This allows for using nonlinear controllers for low-level control, where the training of the artificial neural network is done by reinforcement learning using data collected by the motor controller. However, there remains the problem of how one can efficiently send the data collected by the motor controller to the main computer.

2.7 Ethernet

Low-level controllers run at high frequency so naturally they generate a lot of data. If the low-level controller is an artificial neural network, the data can be used to train the artificial neural network. Sending large amounts of data off the motor controller to the main computer requires a protocol and hardware that exceeds what serial protocols can provide. This is because there is considerable distance between the motor controller, of which is near the legs of the quadruped, and the main computer, of which is near the center of the quadruped. Although the distance between the legs of the quadruped and center of the quadruped might seem small physically, this is not the case electronically. This is due to the fact that data between the motor controller and main computer is sent at a high bitrate, and at high bitrate even short distances appears “large” to the electronics.
To address the problem of needing to send large amounts of data at a large bitrate, we can turn to ethernet and fiber optics for transmitting the data collected by the motor controller to the main computer. Fiber optics allows for one gigabit per second of data transfer through the wire and the ethernet protocol can be used to allow the motor controller to communicate with the main computer. The ethernet protocol can also be full duplex so the main computer can send commands to the motor controller while the motor controller sends collected data to the main computer.

Crucial hardware for ethernet is the magnetics and RJ45 jack. Magnetics are there to shield the ethernet connections from external noise; RJ45 is what the fiber optic cable connects to.
To summarize, in addition to the microcontroller, sensors, and circuitries (inverter) for motor control, the motor controller for the robotic quadruped will additionally include an FPGA and ethernet hardware (Figure 2.30). The microcontroller will first collect sensor outputs and receive user commands via ethernet. Then, it will combine sensor outputs and user commands and form an “input” into an artificial neural network. This input will be sent to the FPGA via a serial protocol. Having programmed the FPGA become an artificial neural network, the FPGA will take the input and generate an output. This output is sent to the microcontroller via the same serial protocol. After receiving the output from the FPGA, the microcontroller will control the inverter (a circuit that controls the amount of current flowing through the motor) to achieve control over the motor. These input-output pairs will be sent to the main computer, through ethernet, by the microcontroller because the microcontroller.

2.8 Summary

We began the chapter with the central idea of the methodology behind the control of a robotic quadruped: to imitate central pattern generators. For the methodology behind building the quadruped, we first reviewed the basics behind motor control and worked our way to the details of electronics behind motor control. This was followed by a brief introduction to biotensegrity and how it can be applied to leg design. Magnetic skin sensors were then introduced as sensing of the terrain was needed to provide the quadruped a good understanding of the terrain. Having discovered that magnetic skin sensors is not readily applicable to motor control, we introduced ideas from reinforcement learning and artificial neural networks. Finally, we concluded the chapter with an overview of field-programmable gate arrays, ethernet, why they will be added to the motor controller, and how they can integrate with the motor controller. Using these smart motor controllers, the quadruped can adapt its low-level controller as a response to the terrain—simply by changing the weights of the neural network formed by the FPGA. It is the hope that the quadruped can adapt and walk on challenging terrains using these smart motor controllers.
3 Project status and developments

3.1 Introduction
The most important element of this project is the robotic quadruped because the ultimate goal is to enable machine learning for robotic quadrupeds. Therefore, development of the quadruped comes first. For example, while one could build just a leg to test that the machine learned motor controller works, one would need to test it on the actual quadruped before the experiment is an actual success. Hence, the primary focus of this project is placed on building the quadruped over the motor controller.

The process of building an entire quadruped procured a lot of challenges that has not been foresaw during the planning stage, but which has been overcome as they occurred. Unfortunately, these unforeseen challenges shifted the timeline to that beyond the original forecast. Hence, in the remainder of the report, the focus will be changed to the development of the robotic quadruped, challenges that occurred, and how it had been overcame. This report will essentially answer the question: can one build a robotic quadruped the likes of the mini-cheetah at HKU? The reader might be excited to know that the answer is a resounding yes, and this report will serve as a guideline for how one should proceed.

3.2 Overview of building a robotic quadruped
The process of building a robotic quadruped begins with the design of both the motor and mechanical design. This process has to be inter-linked because the motor forms parts of the quadruped hence one should consider how forces should be transmitted between these components. Note that the motor controller is part of the motor design because one needs to consider how the controller fits within the housing of the motor, as well as how the stator would be connected to the controller.

The mechanical design involves the legs of the quadruped, the feet, the body chassis, and the internals of the quadruped—this involves figuring out how the electronics will be placed within the quadruped. The motor design involves deciding how heat should be dissipated from the stator and how loads from the output should be transmitted to the mechanical housing instead of the rotor.

Internal electronic includes the inertial measurement unit, a computer, a receiver if one desires to use an external transmitter to remote control the quadruped, a device to facilitate communication protocol within the quadruped, a method of removing heat from the internal of the quadruped, battery, and power distribution.

After the design of the motors and mechanical structure of the quadruped comes fabrication. Motor controllers are printed circuit boards hence one would generally generate a set of Gerber files and send to a manufacturer to have them made. For the mechanical design, one should find a CNC prototyping service to fabricate the initial parts, and then to manually rework these parts at a local machine shop. This is especially important if resource is limited (i.e. not enough machines available) as a total of twelve motors would need to be fabricated.

Lastly, the internal electronics should be added to the internal chassis. In the remainder of this section, we will go through each of these processes and see how they have been implemented within this project.
3.3 Motor Design

3.3.1 Overview

The motor design can be divided into the stator, rotor, gearbox, and mechanical housing. This design is similar to that of the motors used in the mini-cheetah. The difference lies in using a larger stator, custom rotor, an additional heatsink, and different bearings.

3.3.2 Stator and rotor
The stator chosen is 100mm in diameter and 10mm thick. It was chosen because it has a large air-gap radius which has been shown in the MIT cheetah robots to be capable of generating larger amount of torque than those with low air-gap radius. The rotor was chosen because it is made of silicon steel which is steel with high percentage of silicon. This means that the rotor will have less eddy current due to having lower percentage of metal and will be less resistive to magnetic fields due to higher amounts of silicon. The rotor had to be trimmed in diameter to better integrate with the designed motor housing. An additional usage of the rotor is to contain the magnetic field due to the magnets that would be glued onto the rotor—it is made of silicon steel which has low magnet reluctance, which is to say that magnetic field prefers to flow through it than to flow through air.

The rotor component shown in Figure 3.2 is solely to glue the magnets onto. However, the rotor needs to be attached to the motor housing which is what the part in Figure 3.3 is for. Similar components in other commercial rotors are shown on the right of Figure 3.3. Therefore, unlike the mini-cheetah which uses a rotor that came along with the stator, the rotor in this motor is to be machined and customized. An additional benefit of using a custom rotor is the ability to use stronger magnets. Most commercial rotors use N48 magnets which are grade 48 neodymium magnets because they are strong for how common and cheap they are. There are, however, higher grade neodymium magnets which are not that much more expensive in small scale. By designing our own rotor, one can use these stronger magnets and increase the amount of maximum reachable torque. This is important because there is a limit to how much torque the stator can produce, and that is determined by the amount of copper that one can coil onto the stator. This is obviously limited as there is only so much copper you can coil onto the stator. Consequently, the only other way to increase torque is by using stronger magnets.
Looking at the rotor of commercial rotors, one can observe that the magnets are glued onto the rotor such that they are straight. This is because it is desirable to keep the magnetic flux due to each magnet inline with the slots on the stator. Put simply, if the magnets are not glued on straight and are tilted, the magnetic flux would not be distributed evenly hence the motor would behave differently when the rotor is at different angle because the flux is not evenly distributed. In order to achieve a similar effect on our custom rotor, a part for aligning magnets is added that is to be pressed onto the rotor (shown on the left in Figure 3.4). Finally, the complete rotor design (without the magnets) is shown on the right of Figure 3.4.

3.3.3 Gearbox

Figure 3.20 shows the gearbox located within the motor housing. A gearbox can amplify the amount of torque produced because the output of the gearbox rotates at a slower speed than the input of the gearbox (provided the
gear ratio is greater than 1). Due to the conservation of energy, the decrease in speed is compensated by greater producible torque. Because producible torque is of primary concern, a gearbox is used.

The gearbox is a planetary gearbox and follows the design of the MIT mini cheetah. This is because it has been experimentally proven to work well and the low gear ratio (6:1) allows for quasi-direct drive. A gear ratio of 6:1 means that the output of the gearbox rotates once for every sixth rotation of the input to the gearbox. A quasi-direct drive is just a gearbox with low gear ratio (6:1 is considered low gear ratio as gear ratios can reach the hundreds). Quasi-direct drive is important because backdrivability is increased and backlash is kept low, therefore any changes to the output would be directly reflected to the input—allowing the motor to accurately adjust its force output at the end effector as a response to what the end effector is experiencing.

Intuitively, a high gear ratio is bad for backdrivability because the higher the gear ratio, the higher the reflected inertia. Reflected inertia is the inertia that the rotor perceives as a result of a torque applied onto the output of the gearbox. The higher the inertia, the more sluggish the response; imagine swinging something heavy as opposed to something light. When swinging something heavy, it is much harder to control hence the bandwidth at which one can control the heavy object is low. When swinging something lightweight, it is much easier to control hence the bandwidth at which one can control the lighter object is higher. The same principle applies to the rotor, and the “weight” of the object as perceived by the rotor is proportional to the gear ratio. Hence, higher gear ratio equals lower bandwidth which results in a slower response, whereas lower gear ratio equates to higher bandwidth which results in a quicker response.

Following the mini cheetah, the sun gear is the GEABN0.5-20-8-K-4 from Misumi, planetary gears are GEFHB0.5-40-5-8-W3-H12 from Misumi, ring gear is the KHK SI0.5-100.

3.3.4 Mechanical housing

![Figure 3.6 Motor front housing. Circled in red on the right shows a slot for wiring the stator to the motor controller. All small holes shown contain M3 threads.](image)

Shown in Figure 3.6 is the front of the motor which acts as the front housing of the motor as well as the output of the motor. The leaf-shaped cutouts of the motor is there to allow for heatsinks to pole out. Smaller holes are tapped with M3 threads for fixture. Figure on the right shows the inside of the front housing. There are bosses
sticking out at the center of the housing that the stator, ring gear, and a bearing is meant for pressing onto. These components are shown in Figure 3.7.

![Figure 3.7 Components that held onto the front housing.](image)

The back housing (Figure 3.8) serves the purpose of transferring load to other parts of the body, fixture the rotor onto the motor housing, hold the motor controller, and to fixture itself to the front housing.

![Figure 3.8 Back housing of the motor. Left: the part of the back housing that serves the purpose of fixing the rotor to the motor. Right: an enclosure on the back housing to place the motor controller into. There are screw holes to hold the motor controller circuit board to the motor.](image)

To transfer load, a boss, shown in Figure 3.9, is added which is to be inserted into the front housing.
Figure 3.9 Boss is circled in red on the left. The intent is to insert this circular boss into the front housing such that load experienced by the front housing is transferred through this circular boss to the back housing. Figure on right shows the boss inserted into the front housing.

To fixture the rotor onto the back housing, a bearing is inserted into the back housing, and the bearing holds onto the rotor. This is shown in Figure 3.10.

Figure 3.10 Figure shows the rotor held onto by the back housing via a bearing. The bearing is highlighted in red. Bearing allows for the rotation of the rotor to be independent (approximately) of the back housing. Moreover, bearing removes 5 degrees of freedom and only allows for rotation about its rotational axis. This ensures that the rotor is fixed in place but is allowed to rotate.

The back housing is held to the front housing via screws. Figure 3.11 shows a section view of the assembled motor.
Not shown in this view are dowel pins that hold the gears. Because the gears need to rotate about an axis, a pin is needed for the gears to be mounted onto. Dowel pins hardened to HRC58 are chosen for the gears to be mounted onto. Figure 3.12 shows the placement of the pins (drawn in red) such that the gears can rotate.

Figure 3.11 Section view of the motor.

Figure 3.12 Drawn in red are the axis along which the dowel pins are inserted. These pins go through the gears.
Finally, to protect the motor controller, a cover shown in Figure 3.13 is mounted onto the back of the back housing.

*Figure 3.13 Cover. The figure on left shows the side of the cover that faces the motor controller. Some of the bosses (those on the outer edge) are there to transmit load, others are to heatsink components on the motor controller. The figure on right shows the other side of the cover. Holes are drilled for inserting screws (to screw the cover to the back housing).*
3.4 Leg design
3.4.1 Overview

The leg design involves designing how the motors are attached to one another (motors form the hip), the knee for transmitting torque to the leg, the leg itself, and the feet. Different from the mini cheetah, this design does not strive to be as lightweight as feasible. Instead, emphasis is placed onto increasing control bandwidth instead of simplifying control with lighter legs.

Without better material, leg mass does not scale well with volume because given a fixed density, mass and volume is proportional. However, as described by the square-cube law, as a shape grows in size its volume grows faster than its surface area. Therefore, as the robot increases in size (and the leg correspondingly), it becomes impossible to make a “lightweight” leg. This is partly why the mini cheetah is so small—it needs to be small in order for the legs to be light (assuming aluminum legs).

A major advantage of using lighter legs is that one can essentially ignore the inertia effects of the leg. This makes control easier because one does have to account for inertia effects during planning and makes control faster because inertia term involves trigonometric quantities, the calculation of which is usually done on computers with Taylor series expansion (or with lookup tables at the sacrifice of accuracy). These methods are slower compared with floating point operation because there are hardware specific to floating point multiplications, additions, etc.
However, this quadruped is designed for complicated control methodology and functions like trigonometric functions cannot be avoided. In addition, the size of the quadruped has to be considerably larger than the mini cheetah in order to contain more compute and a bigger battery. Therefore, optimizing leg mass is infeasible. Instead, the leg’s bandwidth is optimized so as allow for better responses to control commands. This is the primary design goal for the legs.

3.4.2 Hip design

Hip design involves figuring out how to join three motors together in serial fashion to form a joint with pitch and roll axis, and a link to serially actuate the leg joint.

The first linkage in this serial chain is the joint shown in Figure 3.16.
The joint is very lightweight, and the idea is to use the motor’s housing as structure for the leg. Hence, the joint is merely there to connect motors together. Countersinks (the part nears the holes that sink into the material) are there to place the head of countersunk screws into. This ensures that the motor does not collide with the screws.

The second joint along the serial chain is shown in Figure 3.17.
Akin to the first joint, the second joint contain holes for screws and for dowel pins. The dowel pins is used for transferring load.
The outer-shell of the knee is formed by two parts: the bottom shell and the top shell. See Figure 3.19.
The knee serves primarily as a method to transfer the rotational movement of the motor on the hip to the leg. This is done with two sprockets and a belt contained within the two shells that form the knee (shown in Figure 3.20). The two shells are joined together via the screw holes. Each screw hole has a boss to align the two shells and transmit load.
The design of belts and sprockets is what determines the bandwidth of the leg because the belt is the least stiff component in this system. Thus, the belt is the limiting factor with regards to bandwidth. To increase the bandwidth, the belt needs to be stiffer. An alternative to belt is the four-bar linkage which is essentially a form of transmission whereby solid links are used. Therefore, four-bar linkage allows for much higher bandwidth than a belt because it is stiffer. A belt is preferable to a four-bar linkage in this design, however, because the four-bar linkage would increase the size of the quadruped further, and the size is already considerably larger than other smaller quadrupeds. Also, a belt allows for larger range of motion than a four-bar linkage. Another method of transmission is a cable drive which has the advantage of zero backlash, but to gear-up on a cable drive (to achieve higher gear ratio) requires a large axle (a circular component that a belt is wound around) and would, likewise, take up additional space. Overall, a belt is the most compact form of transmission hence it has been chosen.

The difference between this belt and the mini cheetah lies in the usage of a wider and lower pitch belt. The mini cheetah uses a belt with 1cm width as opposed to the belt used in this design with has a width of 2.5cm. A wider belt allows for higher bandwidth because it is stiffer. In addition, the belt in this design has a pitch of five which is less than that of the mini cheetah. Lower pitch is better for torque because there is more surface for the sprockets to grab onto the belt. Both belts use aramid tensile cords which are Kevlar fibers and are very stiff. A potential alternative are carbon fiber cords which are stiffer but are more expensive.

The specific sprocket chosen is the GPB30MR5250-A-HUF-3 and the GPB40MR5250-A-HUF-3. These sprockets give a gear ratio of 1.3333 and are sized specifically for belts with 2.5cm width. In addition, these sprockets have a profile that is compatible with the chosen belt: 535-5MGT-25 powergrip belts.
The first sprocket is screwed onto the output of the motor as shown in Figure 3.21.

![Figure 3.21 First sprocket in the serial chain. Holes are drilled into the (Luminous Audio, n.d.) sprocket for screws and dowel pins.](image)

The second sprocket is held by the knee using two tapered roller bearings, a bearing spacer, and an axle, shown in Figure 3.22.
Tapered roller bearings are chosen because they are good in a back-to-back fashion with a bearing spacer in-between. The bearing spacer applies tensions in-between the bearings to stiffen the bearings. The axle holds everything together by acting as a clamp between the top and bottom shell of the knee (see Figure 3.23).
The specific tapered roller bearing used is the Timken 30203.

Belt drive requires tensioning, and to tension the belt, a pair of tensioners are used. A pair of tensioners is necessary because the belt would rotate in both clockwise and counterclockwise, hence either side of the belt have the potential to become slack. The tensioners are shown in Figure 3.24
Figure 3.24 A pair of tensioner arms presses onto the belt via two set screws.

The bearings that form the tensioners are the HK0306 drawn cup needle roller bearings riding on 3mm dowel pins.
3.4.4 Leg design

The leg is designed to sustain forces that the ball-point feet will experience and transmit it to the upper body of the quadruped. To actuate the leg, the leg is attached to the sprocket (as shown in Figure 3.24.1) which allows the belt drive to actuate the leg. To secure the leg to the sprocket, holes are drilled onto the side of the sprocket and the corresponding holes are located on the leg. M4 threads are tapped into the holes in the sprocket and countersunk screws are used to bolt the legs onto the sprocket. See Figure 3.24.2.

Figure 3.24.1 Leg design.

Figure 3.24.2 Screw leg onto sprocket.
It is desirable for the leg to be light, and a way to reduce mass as well as to resist bending moments is the I-beam. The I-beam is smaller closer to the feet and increases in size the further it is away from the feet. This is to ensure that it can sustain the bending moments induced by the feet’s point of contact with the terrain (which increases the further away from the point of contact).

The leg is shaped with the curve shown in Figure 3.24.3 because it allows for the leg to be folded close to the knee as shown in Figure 3.24.4. The arch towards the ground near the end of the feet ensures that even when the legs are folded close to the knee, the quadruped can crawl and enables the quadruped to navigate small, constrained, spaces.
The feet design principles have been introduced in Benjamin Katz’s thesis on the mini-cheetah:

1. Grip, so the coefficient of friction between feet and potential terrain has to be high
2. Wide range of contact with ground
3. Damping, so the energy during impacts should be absorbed
4. Durable

For the mini cheetah, squash balls were used as they fulfill all these qualities. In fact, squash balls are used in a lot of 3D printing and acoustic setup because they are very effective at absorbing vibrations. Following this guideline and exploring the 3D printing / acoustic community, a material has been found that is more advantageous than squash balls in terms of these four criteria: Sorbothane.
Sorbothane has friction coefficients comparable with squash balls and is, from the community’s experience, better than squash balls in terms of damping vibrations. Sorbothane dissipates energy by converting them into heat, and are built to be durable as its built to be under constant pressure.

However, Sorbothane are sold in hemispheres, but a sphere is preferable to fulfill the second criteria. Therefore, the feet is designed by gluing together two Sorbothane hemispheres. To mount it to the leg, a “bone” is inserted in-between the two hemispheres and one of the hemispheres has a cut-out to allow the leg to be plugged into.

Figure 3.27 illustrates the leg assembly. Finally, the other Sorbothane hemisphere is glued to the bottom of the assembly to form the entire feet, shown in Figure 3.28.
3.5 Body design

The last to be designed aspect of the quadruped is the body to which the legs are attached to. The guiding principle behind the body is the following:

1. it should be wide enough to contain the width of two motors, and for there to be enough space between the two motors
2. the span of the body should be long enough such that the front legs can be folded backward without colliding with the back legs
3. just tall enough to contain the motors
Following this guideline, the body’s design can be broken into two parts: two mounts for mounting the motor onto, and a chassis that connects these two mounts together. See Figure 3.30.

![Figure 3.30 Top left: body without four legs. Top right: Top view of the body without legs. Bottom left: Motor mount for two motors. Bottom right: Chassis to join two motor mounts.](image)

The motor mount is designed such that the motors form parts of the motor mount. Small holes are drilled in for placing screws, and a giant hole is there to allow wires to escape the chassis and into the motors. The chassis is a 1.5mm thick aluminum sheet bent into a shape that aligns with the motor mounts. There are fins cut-out of the chassis to facilitate heat flow—fans will be placed underneath those fins.

With this design, there are plenty of room for battery, computers, and other necessities.

### 3.6 Quadruped fabrication overview

Once the quadruped has been designed, the next step is to fabricate it. When designing the quadruped, it has been assumed that the machine shop at HKU’s Innowing would contain the sufficient equipment to handle the job. However, the reality was far from it, and challenges related to geography (components sourcing in Hong Kong), tooling availability (in Hong Kong), and other problems one only encounters when it comes to building the quadruped appeared.

All of the encountered challenges, that will be described, had been overcame. However, it came at the cost of shifting the timeline of the project to that beyond initial forecast. Regardless, being one of the first to attempt something akin to this at HKU, this cannot be avoided. Thankfully, this report now serves as a guideline to how one could go about building a robotic quadruped at HKU, because it is indeed possible as this section will proceed to show.
3.7 Motor fabrication

3.7.1 Overview
Six components constitute the motor: front housing, back housing, rotor, teeth, gearbox, and cover. The initial strategy was to use cheap CNC services to fabricate the initial components for the front housing, back housing, rotor, teeth, and gearbox. Then, these components will be reworked at Innowing to assemble the motor. The cover will be fabricated with a CNC milling machine at Innowing. This strategy was chosen because there is only one CNC milling machine at Innowing and it is a machine dated back in the early 2000. Thus, the machine is far weaker than conventional milling machine nowadays. As a result, its maximum RPM is 3800 which is far lower than the usual 12000 RPM. This effectively means that the machine is far slower than other milling machines when milling out parts.

Motor fabrication can be broken down into the following sections:

1. Rework the front housing, back housing, rotor, teeth, and gearbox which are initially made with cheap CNC services (hence the need for reworking the parts)
2. Use the CNC milling machine to develop the cover
3. Sourcing the gears, bearings, rotor, stator, and magnets for the rotor
4. Assemble the rotor and stator
5. Assemble the motor

3.7.2 CNC service and reworking components
To find cheap CNC services, one could look through Taobao and ask for a quote. CNC service provider accepts 3D STEP files when quoting. After inquiring with multiple providers, simply choose the cheapest one that gives acceptable machining tolerance.

Cheap CNC services have loose tolerances so while even the worst CNC machines have amazing tolerance compared to something man-made (the average CNC machine could have up to plus or minus 0.2mm tolerance), a tolerance of 0.2mm is insufficient when making press fits which requires much lower tolerances (perhaps 0.01mm). Therefore, after receiving the parts made by these CNC service providers, they need to be reworked with the CNC milling machine or the lathe at Innowing. For example, say a bearing of exactly 60mm diameter needs to be placed within a hole located in the front housing. If the hole is machined to 60.2mm diameter due to the tolerance, the bearing would slip through the hole which is undesirable since it needs to be fixed in place. If, however, the hole is machined to 59.8mm in diameter, the hole would be too small to press the bearing into. Hence, rework is necessary.

Figure 3.31 shows the components that are machined by the CNC service provider.

![Components](image-url)
To rework these components, the lathe is used. The lathe, shown in Figure 3.32, is a tool used for cutting circular objects. It can be used for enlarging holes or reducing the diameter of a shaft.

Recall that the gears and bearings need to be pressed onto the components (front housing, back housing, etc). But due to manufacturing tolerances, the components needs to be reworked. However, the lathe is a machine that cuts hence it can only be used to remove material. Therefore, to ensure that the components can be reworked, the CNC service provider has been instructed to make the components such that they are reworkable. For example, if a hole needs to be at 60mm, the CNC manufacturers would make the hole as if it was 59.8mm. So, at worst case due
to a 0.2mm manufacturing tolerance, the hole’s diameter would be at 59.6mm or 60mm. If it was at 59.6mm, the lathe can be used to enlarge the hole because the process of making a hole larger is a cutting process which a lathe is suitable for. If instead the hole was manufactured to 60mm, then no rework is needed. It is by using this idea and the lathe that the components are reworked, as shown in Figure 3.33.

![Figure 3.33 Reworking the components at Innowing.](image)

### 3.7.3 Milling parts with the CNC

A computer numerical control milling machine is a machine that removes material from a stock using an end-mill. An end-mill is a cutter that cuts via a rotational motion. The CNC milling machine at Innowing is shown in operation at Figure 3.34.

![Figure 3.34 CNC milling machine at Innowing.](image)

The CNC milling machine can be controlled with GCODE to follow a sequence of instructions to move the end-mill. When the end-mill cuts into the stock, material is removed from it and a shape is formed. So, given a stock (essentially a piece of bare aluminum), GCODE controls the end-mill to remove material from the stock to form a part.
Innowing is relatively new, so being the first to use the GCODE functionality of the CNC milling machine, a lot of experimentation was necessary. This involves figuring out GCODE, how to operate the machine such that it follows the GCODE, configuring the RPM, feed rate, and other CNC related parameters. Figure 3.34 shows the first GCODE program that I ran on the milling machine—it was successful.

Eventually, familiarity with the machine was enough to the point where the cover for the motor can be machined. Figure 3.35 shows the part in the process of being milled. The watery substance are coolant that I manually squeezed onto the stock during the milling process (the machine is in no position for automatic coolant dispensing).
The end result is a cover that fits onto the back housing of the motor, as shown in Figure 3.36.

Figure 3.36 Machined cover fitted onto back housing.

3.7.4 Sourcing gears, stator, rotor, bearings, and magnets

The stator and rotor (silicon steel ring) is found on Taobao. Likewise for the bearings. Gears, however, is difficult to source because it specifically requires Misumi and KHK gears. If one lives in the United States, Misumi gears are easy to acquire because anyone can purchase directly from Misumi’s US online store. Misumi store in other region requires affiliations with a company, and Hong Kong does not itself have a Misumi store (China’s Misumi store requires a Chinese phone number and company affiliation). To acquire the gears needed for the design, the
purchase was made through a middleman, found also on Taobao, who was able to purchase from the Chinese Misumi store and relay the components over to Hong Kong. The KHK ring gear was sourced via a Hong Kong official retailer of KHK gears: Yuen Fat Steel Gear. The ring gears were stocked in Japan and was delivered to Hong Kong to the retailer (see Figure 3.37). The magnets were sourced on Alibaba. There are many magnet manufacturers on Alibaba, and a simple search for N52 magnets was sufficient.

![Figure 3.37 KHK gears.](image)

### 3.7.5 Assembling the stator and rotor

The assembled rotor after the components were reworked is shown in the left of Figure 3.38. The right of Figure 3.38 shows the rotor after the N52 magnets was glued onto the rotor. The gluing of the magnets was done with Loctite 648 which is meant for bonding cylindrical parts (i.e. the rotor).

![Figure 3.38 Assembled rotor without magnets on the left, assembled rotor with magnets on the right.](image)

The magnets was glued in an alternative north and south pole fashion, so the first magnet has the north pole facing inward, the second magnet has the south pole facing inward, and so on.
The stator, shown in Figure 3.39, was coiled by hand. Copper wires with inner diameter (without the enamel) of 1.2mm was used. Copper wires need to be enameled because the wires have to be insulated from one another when coiling, otherwise it would effectively be one giant copper wire, and it is desirable to have as much coils as possible because the flux produced by the stator is proportional to the number of coils.

While using thinner copper wires will lead to more coils, the amount of current one can pulse through the wires will be less as otherwise the wires will heat up to the point where the enamel on the wires melt, and the wires will be shorted to one another. Therefore, there is a tradeoff between the thickness of the copper wire and number of coils. Hence, the critical factor in determining torque production is the amount of copper one can fit onto the stator. As shown on the right of Figure 3.39, good amount of copper fill is achieved by using thicker copper wires compared to commercial motors. Commercial motors use thinner copper wires because these are easier to coil by machine, but the amount of copper fill achieved with this method is less than if you hand coil the motors.

The stator is coiled in a wye-connection scheme in which the three phase of the motor is shorted at a floating neutral point. I used a winding scheme calculator, shown in Figure 3.40, to assist with the coiling.
3.7.6 Assembling the motor

Once the magnets are glued to the rotor and the back housing has been reworked, the rotor can then be retained onto the rotor by a bearing that lies between the rotor and back housing. See Figure 3.41.

Similarly, the stator, ring gear, and bearings can be pressed onto the front housing once the rework is completed. See Figure 3.42.
When assembling the motor, it was discovered that there exists an alignment problem between the rotor and the front housing of the motor. This is because initially, the 3-jaw chuck (right of Figure 3.41) on the lathe machine at Innowing was non-operationally and a 4-jaw chuck (left of Figure 3.41) was used instead. The problem with using a 4-jaw chuck is that the center is not automatically aligned, as in when an object is grasped by the jaws (the part of the chuck that is grasping onto the object), the center of the object will not be aligned with the center of the lathe. Thus, the user has to be manually tune the position of the object within the 4-jaw chuck such that the object is centered. This is different to that of a 3-jaw chuck, of which automatically aligns the object to the center of the lathe.
When I initially reworked the components in the 4-jaw chuck, there were misalignments due to the process of manually aligning the center of the object with that of the lathe. This caused the center of the reworked motor components to become misaligned, and the assembled motor (Figure 3.44), whilst functional, has rotational angles where the gears would press into one another, causing cogging and friction that renders the motor unusable.
Eventually, the 4-jaw chuck was replaced with the 3-jaw chuck which automatically aligns the center of the component to the center of the lathe. To fix the misalignment errors, the rotor and back housing was reworked in the 3-jaw chuck. See Figure 3.45.

During the process of reworking both the rotor and back housing such that their centers are aligned with each other as well as with the front housing, the design of the interconnect between the rotor and back housing was changed to accommodate a smaller bearing. Figure 3.46 illustrates the difference. The change was enacted
because it was found that a smaller bearing is better suited due to lower friction hence less loss due to energy loss to acoustic and heat (this is partly why larger bearings are noisier).

Figure 3.46 Left: Before rotor rework. Right: After rotor rework.

With these changes and re-alignment of centers, the assembled motor has smoother rotations and good backdriveability: they are now fit for usage.

3.8 Leg fabrication

3.8.1 Overview

The leg is comprised of the top shell, bottom shell, leg, feet, sprockets, belts, axle, bearings, bearing spacer, and tensioners. Manufacturing of the top shell and bottom shell has to be done by the CNC service provider as the CNC milling machine at Innowing would take too long for how large and meticulous the parts are. Thankfully, the tolerance for the top, bottom shell, and feet are less demanding and would require no rework. The sprockets are ordered from Misumi and are reworked with the CNC milling machine to drill a set of holes that will be used to secure onto the legs of the quadruped. The axle and bearing spacer are machined out of the CNC milling machine at Innowing and cleaned-up with the lathe. The tensioners are cut out of 10mm thick 6061 aluminum sheets with the waterjet cutting machine at Innowing.

3.8.2 Sprockets rework

The sprockets have to be reworked because the leg and motor has to be attached to the sprockets. To rework the sprockets, the CNC milling machine is used. Firstly, a fixture for holding the sprockets, shown in Figure 3.47, was made with the CNC milling machine.
Next, the sprockets, clamped onto by the fixture, was placed onto the CNC mill and a series of holes was cut into the sprockets. See Figure 3.48 for the reworking process. See Figure 3.49 for the reworked sprockets.

Figure 3.47 Fixture for clamping onto the sprockets.

Figure 3.48 Left: the surface of a sprocket is being reworked. Right: holes are being drilled into a sprocket.
Finally, threads were tapped into the holes with a tap (Figure 3.50).

3.8.3 Top and bottom shell
The top and bottom shell were also machined by outsourcing. The design was such that it could be completed with a 3-axis milling setup so as to keep cost down. While it could have been done with the milling machine at Innowing, it would have taken too long to mill all four shells. See Figure 3.51 for the outsourced components.
There was no rework required for the top and bottom shells. Figure 3.52 shows the reworked sprockets mounted on the bottom shell.
Figure 3.52 The top figure shows the bottom shell with sprockets, belt, and leg mounted on. The bottom shell itself is mounted onto the front housing of the motor. The bottom figure shows the same but with the top shell mounted on.

3.8.4 Leg assembly
The outsourced leg component and its assembly with the reworked sprocket can be seen in Figure 3.53.
3.8.5 Feet assembly
The feet is made from three components: Sorbothane hemisphere, reworked Sorbothane hemisphere, and a “bone”. The Sorbothane hemisphere was purchased from PeakHifi as the ones they sell have a hardness of 70 shore which is the highest on the shore hardness scale in terms of Sorbothane. The Sorbothane was reworked by first milling a hemisphere into a fixture using the CNC milling machine at Innowing (see Figure 3.54)
To rework, the Sorbothane hemisphere, the hemisphere was squeezed into the fixture and a set of Forstner drill bits was used to drill holes into the Sorbothane. See Figure 3.55.

The “bone”, shown in Figure 3.55.1, is outsourced, and is made with electrical discharge machining. Each piece costs 200HKD and is to be inserted into the reworked Sorbothane.
The full assembly can be seen in Figure 3.55.2.

3.9 Body assembly
Due to the aforementioned delays, the body was not fully assembled prior to the report. However, a rudimentary version has been pieced together as seen in Figure 3.56.
3.10 Rudimentary motor controller development

3.10.1 Building a Motor Rig

To test a control algorithm on a motor, it is necessary to build something for clamping the motor in place and to align the hall-effect encoder with the center of the rotor.

The beam and metal plates are made of aluminum because they are soft enough to cut with conventional power tools. A drawing is drawn on the aluminum plates and manually drilled.
The parts are assembled with T-shaped screws and flanged nuts. T-shaped screws are screws shaped like the letter “T” and flanged nuts are hex-shaped nuts with a washer attached (see Figure 3.58). A wooden block containing a magnet is attached to the shaft of the motor to be sensed by the magnetic encoder. A cheap motor is used temporarily to assist the development of the motor control algorithm.

3.10.2 Reading from a Hall Effect Encoder
To test the hall-effect encoder, the encoder chip is connected to a breakout board—a printed circuit board used explicitly for testing electronic devices—and connected to a microcontroller. The microcontroller is placed on a development board—a printed circuit board that holds a microcontroller and facilitates rapid prototyping.
The microcontroller is programmed with embedded C and the encoder communicates with the microcontroller via the SPI protocol.

3.10.3 Testing the current sensor
A current sensor is used to inform the motor controller of the current flowing through the motor.

![Figure 3.60 Left: Zoomed out view of current sensing setup. Right: Zoomed in view of current sensor.](image)

Current flows through the sensor and the readings are sent to the microcontroller. Code has been developed to perform the reading.

3.10.4 Soldering components for the motor controller
Solder is a metal that melts at high temperatures, usually two-hundred and seventy degrees Celsius, and is used to connect electronic components. Solder is usually made to look like wires and to apply solder, the wire of solder is held to the components to be joined and melted. Some special components require the use of solder paste, a variant of solder that is in paste form and contained within a tube as shown in Figure 3.61.
The solder paste is applied to solder components that are hard to solder with normal solder.

3.10.5 Testing the Mosfets
Mosfets are electronic switches that can be used to control the amount of current flowing through a motor. A microcontroller can be used to switch the Mosfet on and off. When the Mosfet is on, current flows through the Mosfet, otherwise no current flow.
Mosfets are particularly sensitive to static shock because the static shock has high voltage and the films within the Mosfet cannot sustain high voltage. Hence, extra care (using an antistatic wrist strap, properly grounding oneself) was taken to before handling such components.

3.10.6 Prototyping a motor controller on breadboard

A circuit to control the motor is wired on a breadboard to develop the motor control algorithm. The circuit consists of the tested components.

Figure 3.62 Mosfet wired for testing.

Figure 3.63 Prototype motor controller. The circuit contains six Mosfets (black housing with silver back component) which act as electronic switches to control the flow of current into the motor. In-between the Mosfets are current sensors (blue base, black housing components) that sense current and send that information back to the microcontroller. Some capacitors (circular silver component, bottom left) supply the motor with extra current when demand is too large and sudden to be supplied by the battery (although the battery is physically close to the motor in terms of human standards, electronics run at a much higher frequency so even a small wiring distance between the battery and motor could be large depending on the frequency).
The code for controlling the motor controller, which in turn controls the motor, has been developed.

![Code for the motor controller.](image)

**Figure 3.64** Code for the motor controller.

Upon running the code on the motor controller, the motor spun at a predetermined velocity. Ramping up the velocity caused theMosfets and an integrated circuit to burn.

![Burned integrated circuit. Note the hole near the middle.](image)

**Figure 3.65** Burned integrated circuit. Note the hole near the middle.

A likely cause is the inability of the prototype to handle currents with high frequency. The current prototype was meant for developing the control algorithm and was crudely wired – thick, long wires were used and a problem with building circuits for high frequency using these wires was identified in section 2.3.5 (Printed Circuit Board). The solution, discussed in the same section, is to use a printed circuit board for the next prototype.
3.11 Motor Controller Design (printed circuit board variant)

3.11.1 Overview

![Figure 3.66 Final motor controller.](image)

The motor controller is one of few components that will make or break a robotic quadruped. It needs to be powerful and robust in order to ensure that the robotic quadruped gets enough power to achieve its tasks. Additionally, a motor controller can incorporate different communication protocols, or hardware that can facilitate its compute to perform more complicated tasks.

This project’s initial goal was to develop a motor controller that includes a field-programmable gate array to allow an artificial neural network to run on the motor controller. The idea was that a robotic quadruped can benefit greatly from the “intelligence” brought about by the neural network. However, due to a lot of unforeseen challenges, it was quickly realized that it would not be feasible to assemble and test such a motor controller due to time constraints—because doing so requires the quadruped to be built in the first place, as neural network requires experience to learn, so any experiments performed with neural networks and reinforcement learning needs to be done in the condition that one expects the quadruped to work in. Therefore, even though the motor controller that contains the FPGA has already been designed, the focus was shifted to physically develop the motor controller without the FPGA first.

In this section, design details of the motor controller and all its design iterations will be detailed.
3.11.2 Designing the circuit and printed circuit board

Before designing a printed circuit board (PCB), a schematic must be drawn so that the PCB software knows which components are connected. A schematic is shown in Figure 3.67 that captures the process of schematic design.

The components are placed on a printed circuit board such the schematic is followed, and the layout is optimized for performance. The board was then sent to be manufactured by JLCPCB, the process of which took around one week. Figure 3.68 shows the manufactured printed circuit board.
Manufactured printed circuit board. The pads for soldering electronics are tinned to facilitate soldering. On the left, one can observe pads for the power mosfets and microcontroller. On the right, one can observe pads for the encoder.

The surface on the printed circuit board is made of FR4 fiber glass and has a melting temperature around one hundred and fifty-five degrees. Although soldering requires much higher than one hundred and fifty-five degrees, the fiberglass would not be damaged (easily) because heat is distributed towards other parts of the board; heat would not be concentrated at an area, hence with care temperature would not exceed one hundred and fifty-five degrees.

The components were manually soldered to the printed circuit board using solder paste and standard soldering tools. A cheap microscope (Figure 3.70) was used to facilitate the soldering of tiny components. Some components were as small as 1mm in width and 2mm in length.
Having soldered the microcontroller to the printed circuit board, the coded was flashed onto the microcontroller via a serial protocol (SWD protocol). Figure 3.71 shows the code successfully flashed.

Initially, the microcontroller bricked after one successful flash. Turns out that the configuration for the microcontroller’s power was incorrect, which caused the microcontroller to malfunction. This was fixed by changing the boot mode of the microcontroller and flashing in a corrected version of the firmware. Further testing indicated a few problems with the original design and a second prototype was revised. The model for the second prototype is shown in Figure 3.72 and the manufactured board is shown in Figure 3.73.
Assembly of the second prototype confirms that the inverter side of the motor controller is functional. A third prototype is needed to include additional communication protocols and potentially an FPGA.

3.11.3 A third prototype of which includes the FPGA
The original goal was to develop a motor controller PCB with an FPGA and ethernet hence the third prototype has included the functionalities for both.
However, prior to sending out for manufacturing of the third prototype, it was realized that fully developing the motor controller, with all its features with the FPGA, is infeasible. Hence, a simplified version of the motor controller was developed that includes the ethernet feature but discarded the FPGA. This prototype is shown in Figure 3.75.

Ethernet was chosen over Ethercat because Ethercat is meant for real-time systems where the order of messages and timing constraint needs to be strictly followed. This comes at the price of increased delay in order to guarantee the order of messages. Because the focus is on control bandwidth, ethernet is preferable over Ethercat.
The manufactured third prototype is shown on the left in Figure 3.76 and the assembled variant is shown on the right.

Figure 3.76 The components were hand soldered onto the manufactured printed circuit board.

Some pads are not used because they serve as redundant functionalities. A bunch more motor controllers were assembled and are shown in Figure 3.77.
Figure 3.77 A lot more motor controllers were hand soldered.

A rig for testing the motor controller is shown in Figure 3.78
Initially, sparks flew out of the gate driver; further investigation found that extra input capacitance was needed because the ripple voltage was large during motor startup. Without capacitors, the input voltage struggles to remain at a certain voltage level. After adding the capacitors, the motor controller operates as intended and successfully controls the motor via field oriented control.

3.11.4 Motor calibration and commutation

The motor controller is calibrated by aligning the rotor mechanical angle with the electrical angle to find the offset from the mechanical angle to the electrical angle. This procedure is shown in the below code snippet:

```c
setPhaseVoltage(VOLTAGE_SENSOR_ALIGN, 3.0f*E_PI/2.0f);
HAL_Delay(700);
get_encoder_value();
electric_angle_offset = electricalAngle(encoder_value);
```

Commutation of the motor is done using field-oriented control. The specific technique chosen is the space-vector modulation, the code to achieve this is as follows:
void setPhaseVoltage(float uq, float electric_angle) {
    float uout = uq / MAX_VOLTAGE;
    electric_angle = angle_wrap(electric_angle + E_PI/2.0f);

    int sector = floor(electric_angle / (E_PI/3.0f)) + 1;

    float t1 = SQRT3*sin1(sector*(E_PI/3.0f) - electric_angle) * uout;
    float t2 = SQRT3*sin1(electric_angle - (sector-1.0f)*(E_PI/3.0f)) * uout;
    float t0 = 0.0f;

    float ta, tb, tc;

    switch (sector) {
    case 1:
        ta = t1 + t2 + t0/2.0f;
        tb = t2 + t0/2.0f;
        tc = t0/2.0f;
        break;
    case 2:
        ta = t1 + t0/2.0f;
        tb = t1 + t2 + t0/2.0f;
        tc = t0/2.0f;
        break;
    case 3:
        ta = t0/2.0f;
        tb = t1 + t2 + t0/2.0f;
        tc = t2 + t0/2.0f;
        break;
    case 4:
        ta = t0/2.0f;
        tb = t1+ t0/2.0f;
        tc = t1 + t2 + t0/2.0f;
        break;
    case 5:
        ta = t2 + t0/2.0f;
        tb = t0/2.0f;
        tc = t1 + t2 + t0/2.0f;
        break;
    case 6:
        ta = t1 + t2 + t0/2.0f;
        tb = t0/2.0f;
        tc = t1 + t0/2.0f;
        break;
    default:
        ta = 0.0f;
        tb = 0.0f;
        tc = 0.0f;
    }

    ua = ta * MAX_VOLTAGE;
    ub = tb * MAX_VOLTAGE;
    uc = tc * MAX_VOLTAGE;

    setPWM();
}
3.11.5 Ethernet

The ethernet functionality was achieved by including the LAN9250 chip in the design. The LAN9250 chip, shown in Figure 3.80, serves as a link between the ethernet layer and the microcontroller.

![Figure 3.79 Ethernet jack, below of which is the LAN9250 chip.](image)

The LAN9250 receives ethernet frames and stores them in memory. The chip provides a QUAD-SPI interface to read its data; an external microcontroller can treat the LAN9250 as if it was a memory chip and read data from it. The data would be the received ethernet frames. To send out ethernet frames, the microcontroller must write to the LAN9250 as if it was writing to a memory chip. The LAN9250 has a specific address range that is used solely to store ethernet frames to be sent to the ethernet physical layer.

The stm32H725 has an OCTO-SPI interface which is compatible with the QUAD-SPI interface. The following code snippet provide functions to read and write data from and to the LAN9250 chip:
```c
#include "lan9250.h"

static uint8_t generalRxBuffer[10];
static uint8_t generalTxBuffer[4];

void readTestByte(OSPI_HandleTypeDef * hospi, uint8_t * aRxBuffer) {
    static OSPI_RegularCmdTypeDef sCommand;

    sCommand.OperationType = HAL_OSPI_OPTYPE_COMMON_CFG;
    sCommand.FlashId = HAL_OSPI_FLASH_ID_1;

    sCommand.Instruction = 0x03;
    sCommand.InstructionMode = HAL_OSPI_INSTRUCTION_1_LINE;
    sCommand.InstructionSize = HAL_OSPI_INSTRUCTION_8_BITS;
    sCommand.InstructionDtrMode = HAL_OSPI_INSTRUCTION_DTR_DISABLE;

    sCommand.Address = 0x064;
    sCommand.AddressMode = HAL_OSPI_ADDRESS_1_LINE;
    sCommand.AddressSize = HAL_OSPI_ADDRESS_16_BITS;
    sCommand.AddressDtrMode = HAL_OSPI_ADDRESS_DTR_DISABLE;

    sCommand.AlternateBytes = HAL_OSPI_ALTERNATE_BYTES_NONE;

    sCommand.DataMode = HAL_OSPI_DATA_1_LINE;
    sCommand.DataDtrMode = HAL_OSPI_DATA_DTR_DISABLE;
    sCommand.NbData = 4;

    sCommand.DummyCycles = 0;
    sCommand.DQSMode = HAL_OSPI_DQS_DISABLE;
    sCommand.SIOOMode = HAL_OSPI_SIOO_INST_EVERY_CMD;

    if (HAL_OSPI_Command(hospi, &sCommand, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) != HAL_OK) {
        Error_Handler();
    }

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_RESET);

    if (HAL_OSPI_Receive(hospi, aRxBuffer, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) != HAL_OK) {
        Error_Handler();
    }

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_SET);
}
```
void enableQuadSpi(OSPI_HandleTypeDef * hospi) {
    static OSPI_RegularCmdTypeDef sCommand;

    sCommand.OperationType = HAL_OSPI_OPTYPE_COMMON_CFG;
    sCommand.FlashId = HAL_OSPI_FLASH_ID_1;

    sCommand.Instruction = 0x38;
    sCommand.InstructionMode = HAL_OSPI_INSTRUCTION_1_LINE;
    sCommand.InstructionSize = HAL_OSPI_INSTRUCTION_8_BITS;
    sCommand.InstructionDtrMode = HAL_OSPI_INSTRUCTION_DTR_DISABLE;

    sCommand.AddressMode = HAL_OSPI_ADDRESS_NONE;
    sCommand.AlternateBytes = HAL_OSPI_ALTERNATE_BYTES_NONE;

    sCommand.DataMode = HAL_OSPI_DATA_NONE;

    sCommand.DummyCycles = 0;
    sCommand.DQSMode = HAL_OSPI_DQS_DISABLE;
    sCommand.SIOOMode = HAL_OSPI_SIOO_INST_EVERY_CMD;

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_RESET);
    if (HAL_OSPI_Command(hospi, &sCommand, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) != HAL_OK) {
        Error_Handler();
    }

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_SET);
}
void quadSpiWrite(OSPI_HandleTypeDef * hospi, uint8_t * aTxBuffer, uint32_t address) {
    static OSPI_RegularCmdTypeDef sCommand;

    sCommand.OperationType = HAL_OSPI_OPTYPE_COMMON_CFG;
    sCommand.FlashId = HAL_OSPI_FLASH_ID_1;

    sCommand.Instruction = 0x02;
    sCommand.InstructionMode = HAL_OSPI_INSTRUCTION_4_LINES;
    sCommand.InstructionSize = HAL_OSPI_INSTRUCTION_8_BITS;
    sCommand.InstructionDtrMode = HAL_OSPI_INSTRUCTION_DTR_DISABLE;

    sCommand.Address = address;
    sCommand.AddressMode = HAL_OSPI_ADDRESS_4_LINES;
    sCommand.AddressSize = HAL_OSPI_ADDRESS_16_BITS;
    sCommand.AddressDtrMode = HAL_OSPI_ADDRESS_DTR_DISABLE;

    sCommand.AlternateBytes = HAL_OSPI_ALTERNATE_BYTES_NONE;

    sCommand.DataMode = HAL_OSPI_DATA_4_LINES;
    sCommand.DataDtrMode = HAL_OSPI_DATA_DTR_DISABLE;
    sCommand.NbData = 4*sizeof(aTxBuffer) / sizeof(uint8_t);

    sCommand.DummyCycles = 0;
    sCommand.DQSMode = HAL_OSPI_DQS_DISABLE;
    sCommand.SIOOMode = HAL_OSPI_SIOO_INST_EVERY_CMD;

    if (HAL_OSPI_Command(hospi, &sCommand, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) != HAL_OK) {
        Error_Handler();
    }

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_RESET);

    if (HAL_OSPI_Transmit(hospi, aTxBuffer, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) != HAL_OK) {
        Error_Handler();
    }

    HAL_GPIO_WritePin(OCTOSPI_CS_GPIO_Port, OCTOSPI_CS_Pin, GPIO_PIN_SET);
}
void setMacAddress(OSPI_HandleTypeDef * hospi, uint8_t * MAC_ADDR_LOW, uint8_t * MAC_ADDR_HIGH) {
    quadSpiWrite(hospi, MAC_ADDR_LOW, 0x150);
    HAL_Delay(1);
    quadSpiWrite(hospi, MAC_ADDR_HIGH, 0x14C);
}

uint16_t getTxFifoFreeSpace(OSPI_HandleTypeDef * hospi) {
    quadSpiFastRead(hospi, generalRxBuffer, 0x80);
    return (generalRxBuffer[1] << 8 | generalRxBuffer[0]);
}

uint16_t getRxFifoFreeSpace(OSPI_HandleTypeDef * hospi) {
    quadSpiFastRead(hospi, generalRxBuffer, 0x7C);
    return (generalRxBuffer[1] << 8 | generalRxBuffer[0]);
}

void sendTxCommandA(OSPI_HandleTypeDef * hospi,
    int IOC, int FS, int LS, int buffer_size) {
    /*
    * IOC = Interrupt on completion
    * FS = First segment
    * LS = Last segment
    * buffer_size = in butes
    */
    uint32_t tmp = 0 | (IOC << 31) | (FS << 13) | (LS << 12) | buffer_size;
    generalTxBuffer[0] = tmp;
    generalTxBuffer[1] = tmp >> 8;
    generalTxBuffer[2] = tmp >> 16;
    generalTxBuffer[3] = tmp >> 24;
    quadSpiWrite(hospi, generalTxBuffer, 0x20);
}

void sendTxCommandB(OSPI_HandleTypeDef * hospi,
    uint16_t packet_tag, int CHECKSUM, int ADD_CRC_DIS,
    int disable_padding, uint16_t packet_length) {
    /*
    * CHECKSUM = Enable CHECKSUM
    * ADD_CRC_DIS = Add CRC disable
    */
    uint32_t tmp = 0 | (packet_tag << 15) | (CHECKSUM << 13) | (ADD_CRC_DIS << 12) | (disable_padding << 11) | packet_length;
    generalTxBuffer[0] = tmp;
    generalTxBuffer[1] = tmp >> 8;
    generalTxBuffer[2] = tmp >> 16;
    generalTxBuffer[3] = tmp >> 24;
    quadSpiWrite(hospi, generalTxBuffer, 0x30);
}
void setMacControlRegister(OSPI_HandleTypeDef * hospi, int RXALL, int HMAC_EEE_ENABLE, int RCVOWN, int LOOPBK, int FDPX, int MCPAS, int PRMS, int INVFILT, int PASSBAD, int HO, int HPFILT, int BCAST, int DISRTY, int PADSTR, int BOLMT, int DFCHK, int TXEN, int RXEN) {
    generalTxBuffer[3] = (RXALL << 7) | (HMAC_EEE_ENABLE << 1);
    generalTxBuffer[2] = (RCVOWN << 7) | (LOOPBK << 5) | (FDPX << 4) | (MCPAS << 3) | (PRMS << 2) | (INVFILT << 1) | PASSBAD;
    generalTxBuffer[1] = (HO << 7) | (HPFILT << 5) | (BCAST << 3) | (DISRTY << 2) | PADSTR;
    generalTxBuffer[0] = (BOLMT << 6) | (DFCHK << 5) | (TXEN << 3) | (RXEN << 2);
    writeToMacCsr(hospi, 0x01);
}

void writeToMacCsr(OSPI_HandleTypeDef * hospi, uint8_t csr_address) {
    quadSpiWrite(hospi, generalTxBuffer, 0x0A8);
    generalTxBuffer[3] = (1 << 7);
    generalTxBuffer[0] = csr_address;
    quadSpiWrite(hospi, generalTxBuffer, 0x0A4);
}
void setOctospiPrescaler(OSPI_HandleTypeDef * hospi, int prescaler)
{
  OSPIM_CfgTypeDef sOspiManagerCfg = {0};

  hospi->Instance = OCTOSPI1;
  hospi->Init.FifoThreshold = 10;
  hospi->Init.DualQuad = HAL_OSPI_DUALQUAD_DISABLE;
  hospi->Init.MemoryType = HAL_OSPI_MEMTYPE_MICRON;
  hospi->Init.DeviceSize = 9;
  hospi->Init.ChipSelectHighTime = 1;
  hospi->Init.FreeRunningClock = HAL_OSPI_FREERUNCLK_DISABLE;
  hospi->Init.ClockMode = HAL_OSPI_CLOCK_MODE_3;
  hospi->Init.WrapSize = HAL_OSPI_WRAP_NOT_SUPPORTED;
  hospi->Init.ClockPrescaler = prescaler;
  hospi->Init_SAMPLE_SELECTING = HAL_OSPI_SAMPLE_SHIFTING_NONE;
  hospi->Init.DelayHoldQuarterCycle = HAL_OSPI_DHQC_DISABLE;
  hospi->Init.ChipSelectBoundary = 0;
  hospi->Init.ChipSelectHighTime = 0;
  hospi->Init.DelayBlockBypass = HAL_OSPI_DELAY_BLOCK_BYPASSED;
  hospi->Init.MaxTran = 0;
  hospi->Init.Refresh = 0;
  if (HAL_OSPI_Init(hospi) != HAL_OK)
  {
    Error_Handler();
  }
  sOspiManagerCfg.ClkPort = 1;
  sOspiManagerCfg.IOLowPort = HAL_OSPIM_IOPORT_1_LOW;
  if (HAL_OSPIM_Config(hospi, &sOspiManagerCfg, HAL_OSPI_TIMEOUT_DEFAULT_VALUE) !=
    HAL_OK)
  {
    Error_Handler();
  }
}

3.12 Summary

To summarize, the following objectives have been achieved:

1. End-to-end design of a robotic quadruped.
2. Design a motor controller with FPGA and ethernet for reinforcement learning.
3. Fully developed the designed motor.
4. Fully developed the motor controller but without the FPGA.
5. Fully developed the designed leg.
6. Created a preliminary assembly of the body.

Having already achieved these objectives, the quadruped can be assembled by developing all twelve motors, three more legs, and attaching the legs to the partially assembled body—a process in which the leg, along with the motors, are screwed to the mechanical housing. All the machinery used are in Innowing and tooling is readily available from stores both local and online. In light of these achievements, it can be concluded that a robotic quadruped can indeed be built at HKU. Hence, herein lies the achieved objective: experimental result shows that a
robotic quadruped can indeed be fully developed at HKU, and this report serves as a guide for how one could build one.

4 Conclusion
To reiterate, building robots with legs has been a fantasy throughout human history and it is not until now that this fantasy could be made a reality. A robot that can control its legs to walk on challenging terrains is not only just a robot that can walk well but also a sign of artificial intelligence. We have highlighted the important hardware and software components that would be critical if the quadruped were to succeed in walking on challenging terrains and have put together a methodology that aims for end-to-end development of a robotic quadruped that is superior to prior quadrupeds in terms of locomotion.

In the pursuit of developing a robotic quadruped for traversing challenging terrains at HKU, a number of challenges had been encountered and overcome. It is by overcoming these challenges that a motor, motor controller, and legs of the quadruped have been fully developed. Assembly of the quadruped, then, is a mere factor of repeating the procedure until all the motors and legs for the quadruped is assembled, and to attach them to the body.

However, in light of the aforementioned challenges, the focus of the project has shifted from “challenging terrains” to “end-to-end development of a robotic quadruped”. This is a response to unforeseen challenges during the development stage and is a natural occurrence when attempting a problem with an approach that there has been little attempts on.

The developed motor controller contains ethernet which is an advantage over many other commercial controllers available. Although it does not yet contain the field-programmable gate array, the design that includes it is readily available and could be manufactured and tested if so desired.

Future work is to follow the procedure outlined within this report and assemble the quadruped. This could be followed by incorporating the field-programmable gate array and testing how it responds to different terrains.

References


104


