Swol Kat

Major Qualifying Project

Advised By:

PROF. NICHOLAS BERTOZZI
BRADLEY A. MILLER
PROF. ANDREW CLARK
PROF. HAICHONG ZHANG

Written By:

EZEKIEL T. ANDREASSEN
ARJUN GANDHI
MITCHELL R. JACOBS
ANDREW E. MULARONI

WPI

A Major Qualifying Project
WORCESTER POLYTECHNIC INSTITUTE

Submitted to the Faculty of the Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science in Robotics Engineering and Electrical Engineering.

AUGUST 2020 - MAY 2021

Abstract

As the focus in robotics is shifting from specialized robots in controlled environments to generalized robots in unstructured environments, there is greater demand for legged locomotion research platforms. Four-legged robots achieve a balance between speed and stability. In this project, the team designed and fabricated a highly maneuverable robot using 3-DOF legs powered by BLDC motors. This quadruped is an extensible platform for future work in gait development, computer vision, and more.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Current State of Research Quadrupeds

Quadruped development has been popular in industry and academia for over a decade. Until recently most high powered quadrupeds relied upon hydraulic or large electric actuators. One notable historic example includes the General Electric Walking Truck which used hydraulic actuators. This robot was developed in 1968. The robot had a mass of 1300kg and a maximum speed of 5 miles per hour. A more recent hydraulic quadruped is Boston Dynamics BigDog robot developed in 2004. This robot weighed approximately 109 kg and can reach speeds of 2 meters per second (5 miles per hour). BigDog's most notable feature is its ability to carry 50kg of payload on any terrain and up to 154kg of payload on flat terrain [3].

When investigating the early history of electrically driven legged robots it was determined that most required more than 4 legs. This is due to the low speeds at which traditional electric actuators needed to operate at to provide the required torque. One such robot was the Ambler hexapod developed from 1988 to 1991 [3].

Currently, Boston Dynamics's Spot robot represents the most advanced quadruped research. Spot has a 14KG payload capacity, robust dynamics, and the ability to recover from falls. MIT has Mini Cheetah[4], a 20lb robot with a great range of motion, and enough torque to perform a backflip. Stanford has 3 quadruped projects: Doggo, Woofer, and Pupper. Doggo has the highest vertical jumping ability of any quadruped robot produced, weighing in at only 5kg. Woofer is a scaled up version of Pupper, a 12 dof quadruped designed to learn and experiment with walking dynamics. Recently the Open Dynamic Robot Initiative was started by a collaboration of teams from NYU and Germany, The project demonstrates the use of a modular robotic limb in a variety of legged configurations. Each leg module consists of a brushless motor and encoder surrounded by 3d printed enclosure. Most of these research quadrupeds rely on the power density of BLDC actuators to achieve their impressive statistics.

Quadruped research has become a focal point of the robotics industry as the demand for quadrupeds has been rapidly increasing. A main cause of this increase is the introduction of Spot [5] to the industrial marketplace. Quadrupeds are no longer seen solely as research platforms. Instead, they are seen as useful tools to work in environments that are too dangerous for humans. These environments range from power plants and construction sites, all the way to SpaceX's drone ships to inspect landed rocket boosters. These commercial applications continue to grow as more development is accomplished on quadruped platforms.

1.2 Proposed Objectives

The three main objectives for this project are: design a mechanical platform that can be used for future legged robot research at WPI, implement a high-speed control loop to control the actuators of the robot and respond to feedback from the environment, and to implement a dynamic walking gait. The first two objectives

are necessary to build a performant platform to develop the dynamic walking gait and the last objective is the primary goal for this project. The dynamic walking gait occurs when a quadruped robot walks by taking two of its legs off the ground at once, rather than always being statically stable with 3 legs on the ground. In nature, four legged animals usually walk in this dynamic gait because it is more maneuverable than a static gait. Having a quadruped capable of this dynamic walking gait is much more difficult as the robot must balance between two legs during the step, leaving it far less stable. The dynamic walking gait is a goal for many quadruped projects as it is what separates a quadruped that walks in a slow, awkward way from a maneuverable and fast machine.

2 Background

2.1 Quadrupeds at WPI

WPI's research into quadrupeds began in 2011 with the Sabertooth MQP [6]. This project produced a 300lb robot that was cable of carrying a 30lb payload. The team was able to simulate a walking gait for their robot. Since then, there have been 5 other MQP's on quadrupeds. The most notable of these is Small Kat [7], a RC servo based, 16 DOF quadruped, which was capable of a static walking gait. Mechanical limitations in size and motor power prevented them from achieving a dynamic walking gait. WPI has also experimented in some non-traditional areas of quadruped design. The HydroDog [8] project used hydraulic muscles to preform bounding and hopping gaits. Low Cost Quadruped [9] while limited in mobility was able to perform SLAM and facial recognition on the robot. Even with all this research, WPI has yet to produce a stable quadruped platform easily usable for research.

2.2 MIT Mini Cheetah

The MIT Biomimetic Robotics Lab developed Mini Cheetah as a small, robust and inexpensive robot in 2019. Achievements of Mini Cheetah include successfully implementing numerous dynamic gaits at speeds of up to 2.45 meters per second and executing 360 degree backflips. Mini Cheetah is an extension of previous research at MIT that has produced several highly capable quadrupeds. [4].

Mini Cheetah served as an inspiration for this project as it implemented custom BLDC actuators, demonstrating their viability at a similar scale to ours. Mini Cheetah also focused on low inertia limbs, which minimized the dynamic forces acting on the robot. Additionally, Mini Cheetah needed to maintain a 1khz control loop per actuator, which we used as a benchmark when choosing and testing components.

Our project differs from Mini Cheetah in many ways. The cost of actuators alone for Mini Cheetah was $3600. This cost would make the project not viable as an MQP. Instead, our project uses off the shelf components, along with 3D printing and carbon fiber plates to dramatically reduce the cost. Additionally, our

project uses a wider base than that of the Mini Cheetah to create a more stable stance, as well as creating additional space to house the of the shelf components.

2.3 ODrive Motor Controllers

For a long time, hobbyists have had access to high power brushless direct-current (BLDC) motors that are meant for quadcopters, but due to the current offerings of BLDC motor controllers these motors have only been useable for high RPM, continuous rotation tasks. Recently, a motor controller has emerged called ODrive that implements a field-oriented control algorithm on BLDC motors. This allows for the low cost, high power BLDC motors to be controlled like servos. We decided to use this new motor controller on the robot because power density is incredibly important for a quadruped robot.

Field-oriented control is a strategy for controlling brushless motors that uses the magnetic field created by the currents in the motor coils and their interactions with the magnetic field produced by the rotor. Maximum torque for a BLDC motor is produced when the magnetic field produced by the current flowing through the motor coils is perpendicular to the field produced by the rotor. This form of control measures the rotor position and drives current through the correct coils so that maximum torque is produced. Varying the magnitude of the current controls the amount of torque being produced.

2.4 Quadruped Walking Gaits

Dogs and other 4 legged animals use a variety of different gaits based on speed, terrain, maneuverability and energy efficiency [10]. Gaits fall into one of 2 categories. Symmetric Gaits are gaits where the right and left limbs alternate. Asymmetric Gaits are those where the legs move in unison. For this project the team focused on implementing the 3 gaits below.

2.4.1 Intermittent Crawl Gait

In this asymmetric gait 3 legs always remain in contact with the ground. The center of mass of the robot is shifted to keep the center of mass within the triangle of contact. All 4 legs are in contact with the ground while the body is moving forward. As the center of mass of the robot is always within the triangle of contact for the robot this gait is incredibly stable. It also minimizes dynamic loads on the robot as the foot taking the step is generally under little to no load.

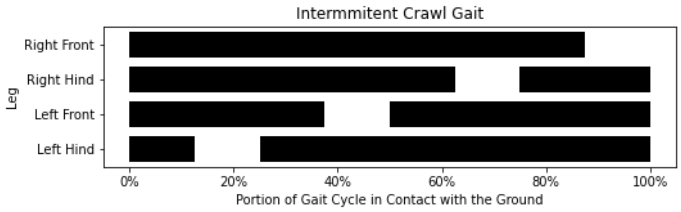


Figure 1: *Foot Contact Patterns for Intermittent Crawl Gait*

2.4.2 Crawl Gait

The asymmetric crawl gait keeps 3 legs in contact in the ground and alternates which leg is in the air at any time. The crawl moves the center of mass in a straight line in the direction of motion. This like the Intermittent Crawl Gait this gait is considered stable as the robots center of mass never leaves the triangle of contact. This gait has the potential to reach a much higher speed than the Intermittent Walking Gait however in doing so this increases the dynamic load on the quadruped.

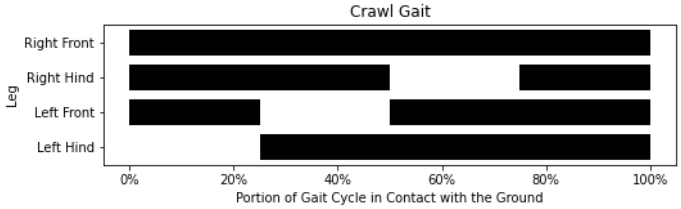


Figure 2: *Foot Contact Patterns for Crawl Gait*

2.4.3 Trot

The Trot Gait is a symmetric gait where the two opposite legs move in unison. This gait unlike the previous two is inherently unstable. When both feet are raised in the air the robot needs to balance on the other two legs. If active balancing is not present the robot needs to step very quickly in order to avoid tipping over. However what this gait sacrifices in stability it gains in speed. The trot gait is significantly faster than both Crawl and Intermittent Crawl and in nature is often used by 4 legged animals to travel long distances quickly.

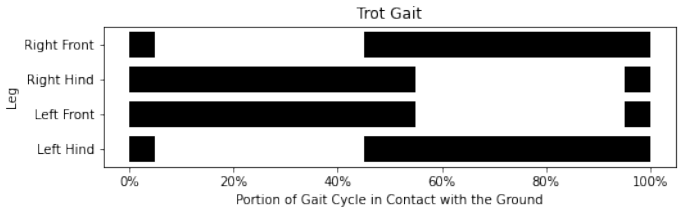


Figure 3: *Foot Contact Patterns for Trot Gait*

3 Methodology

3.1 Engineering Process

Going into this project, we were aware that we were taking on a task that teams have failed to succeed at in the past at WPI. We set out with an ambitious timeline, spreading the work out throughout the entire year. We began by simultaneously developing the basis of the control system and building the first set of leg prototypes. This included steps such as building up the control system to control one motor all the way through controlling multiple ODrives. This allowed a few different leg revisions to be created and tested by hand before the electronics could control an entire leg. Once the electronics were functional, we began testing individual legs and using single legs to develop the control software. Once the design of a single leg was validated, the rest of the quadruped was designed. Simultaneously, a visualization system was created to continue development of the control system. Once the quadruped was assembled the visualized code was ported onto the final quadruped. The robot control system was validated using suspended testing. This testing was followed by lowering the robot to test the mechanical system.

3.2 Motor Selection

Having a high power to weight ratio is critical to create a maneuverable robot. Therefore a motor with a high power output and high peak torque that is in the weight budget is required. Motor power directly contributes to the power to weight ratio, so having a high power output is optimum. A high motor torque output allows for a lower and lighter reduction. With 12 motors needed to build 4 legs, the price of each motor was considered while choosing a motor for the robot.

There are three parameters that are focused on while selecting a motor for this application: the continuous power rating, Kt and Kv. The continuous power rating is an indicator as to how much current can continuously run through the motor without causing overheating or damage. Due to the variation in methods of measuring the continuous power rating, this parameter is only used when comparing motors made by the same manufacturer. The motor Kt and Kv parameters describe how the motor torque and speed relate to the current and voltage being

supplied to it. Kt is the torque constant and in our case has units Nm/A. Kt is multiplied by the current flowing through the motor to obtain the torque applied at any instant. Kv is the motor speed constant and has the units RPM/V. The motor Kv is multiplied by the bus voltage to find the maximum unloaded speed of the motor.

Of the motors that were looked at, the best option appeared to be the 9225 motor from Multistar. Two variants of the motor that are easily obtainable online are the 160Kv and 90Kv motors. The 160Kv had a continuous power rating of 1100W and the 90Kv had a continuous power rating of 900W. The 90Kv motor was more than double the price of the 160Kv motor.

With hobby BLDC motors, the Kt of the motor can estimated using the following equation from the Kv. [11]

$$Kt = 8.27/Kv$$

With two motors that have similar continuous power ratings, the next criteria is the torque output of the motor. The 160Kv motor has an estimated Kt of 0.05168 Nm/A and the 90Kv motor has an estimated Kt of 0.09188 Nm/A. For this robot, the optimal motor is the 90Kv variant of the 9225 motor as it would have a similar power rating with a higher torque constant, meaning less reduction is required. However, due to the higher price of the 90Kv variant the robot was built with the 160Kv motors and a higher gear reduction.

3.3 BLDC Motor Torque Curves

Due to how the ODrive motor controller current limits the motor, the effective torque curve of the BLDC motors used to control the robot differs from the motor torque equation for a motor that is only voltage limited. There are two main regions of the current limited motor torque curve that act differently: the current limited region and the voltage limited region.

During the current limited region, the motor produces a constant torque regardless of motor speed. This happens because the back EMF produced by the rotor is not of a high enough voltage to prevent the desired current from flowing through the coils. Since motor torque is proportional to the current flowing through the coils, a constant torque can be achieved in this region.

In the voltage limited region, the magnitude of the back EMF is high enough to prevent the motor controller from creating enough potential for the desired current to flow through the motor. In this region the torque that can be produced by the motor linearly decreases with speed, following what would happen with a traditional brushed motor.

Due to the low resistance of the BLDC motors being used, a high voltage potential is not needed to create a high current. It is relatively easy for the motor controller to maintain the desired current through the coils. As a result, the motors used in the robot mostly operate in the current limited region of their torque curve. Only rapid movements not in contact with the ground have the potential

to move fast enough to operate in the voltage limited region. In these rapid moves the full motor torque is not needed. This allows us to assume that the maximum motor torque for the robot is that of the current limited region for the mechanical design of the legs.

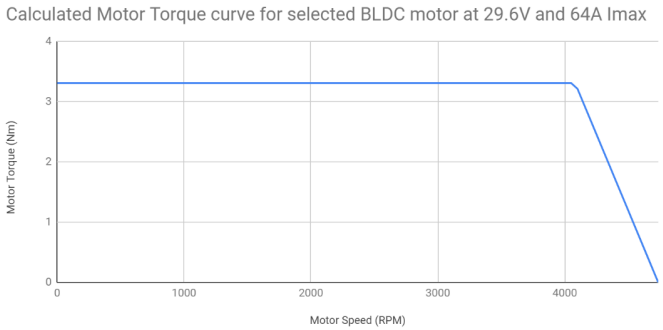


Figure 4: *Calculated torque curve for selected BLDC motor*

3.4 Initial Control System Topology

We decided on a distributed approach for the control system to offload some tasks such as trajectory planning and leg positioning. These tasks are well suited to a microcontroller, rather than an embedded system, as they could be implemented in C++ to run directly on the hardware. After looking at various Arduino and Teensy microcontrollers, it was determined that a Teensy 4.1 has both the processing speeds and GPIO required to connect and control half (3) of the ODrive controllers.[12]

The central controller is responsible for the high-level tasks such as maintaining the balance of the robot and determining setpoints for each leg. Additionally, there should be additional computational capacity, allowing for further development of high level control of the robot. Based upon these requirements, the two main contenders are the Raspberry Pi 4 or the Jetson Nano. The Raspberry Pi 4 and Jetson Nano are very similar, except that the Jetson benefits from CUDA Cores, and although we would not use them, they would have a lot of potential for future projects.[13][14] Ultimately the Raspberry Pi 4 was chosen, as it is the cheaper solution. The pinouts of the Jetson and Pi are identical allowing for the controller to be swapped in the future.

3.5 COVID

While we are satisfied with the progress made throughout the project, there were limitations imposed due to the COVID-19 pandemic. An important consideration was making sure we could continue work should our access to campus be further

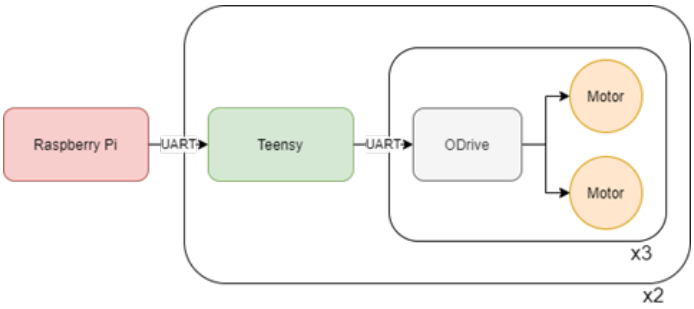


Figure 5: *Initial Control System Topology Diagram*

limited or eliminated. For the most part we were successful in this goal, however there were still many delays caused by the pandemic.

Our timeline kept changing as the ability to work together drastically changed. We were only together working in the same space during a few days throughout the entirety of the project. This meant that communication amongst the team took longer and led to misunderstandings which caused delays on the project.

Additionally, during the middle of B term we were making great progress until most of the team got quarantined for two weeks, halting the progress. Restarting after quarantine was immensely more difficult. Campus had increased restrictions which meant we could no longer meet as a team. This led to handing pieces of equipment back and forth numerous times as we made individual progress, resulting in slower progress overall. These restrictions did not loosen up until the end of winter, at which point the timeline had already drastically shifted. We were only able to be successful by dedicating much of our time to the project.

Throughout the project there were opportunities to switch manufacturing processes to use the machining capability present on campus. Ultimately it was determined not to make these changes. If the project relied upon campus for manufacturing capability, the viability of the project would become entirely dependent upon access to campus. Instead, the project continued to use 3D printing and purchased carbon fiber plates, as the resources to manufacture these materials was independent to any COVID restrictions.

4 Design and Development of Single Leg

4.1 System Overview

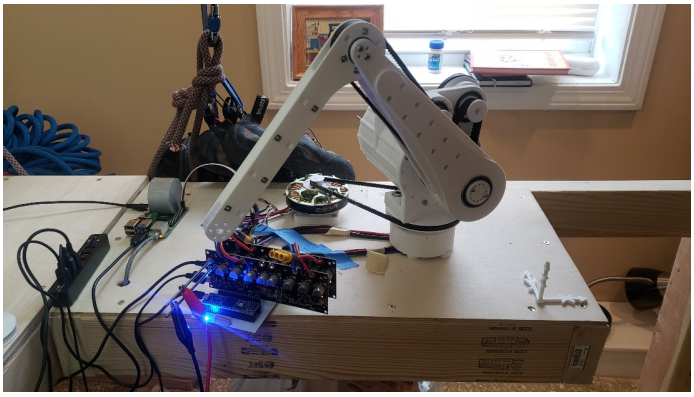


Figure 6: *Full Single Leg Development System*

4.2 Mechanical

Swol Kat has 3 DOF legs. The robot requires reductions that are easily backdrivable and efficient so that the torque applied by the motor is a good representation of the joint torque in the arm. The legs mass and moment of inertia (MOI) must be kept to a minimum to minimize the impact of the leg movement on the center of gravity (CG). This allows for the controls system to ignore the moving masses.

4.2.1 Extremity Design

The extremity is composed of three assemblies: the shoulder, the upper extremity and the lower extremity. To reduce power requirements and increase the speed of rapids when the leg is not loaded, the mass and moment of inertia around the axis of rotation must be minimized.

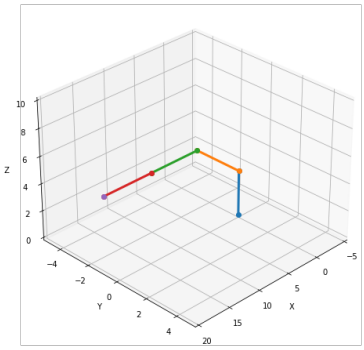


Figure 7: *Upper and Lower Extremity are the Green and Red Segments Respectively*

4.2.1.1 First Design

The first leg design of Swol Kat was done prior to deciding the extremity lengths or the necessary reductions. The first leg served to give the team a weight estimate of the leg, give leg design experience to the team, and get a minimum viable product in the hands of the electrical and programming teams.

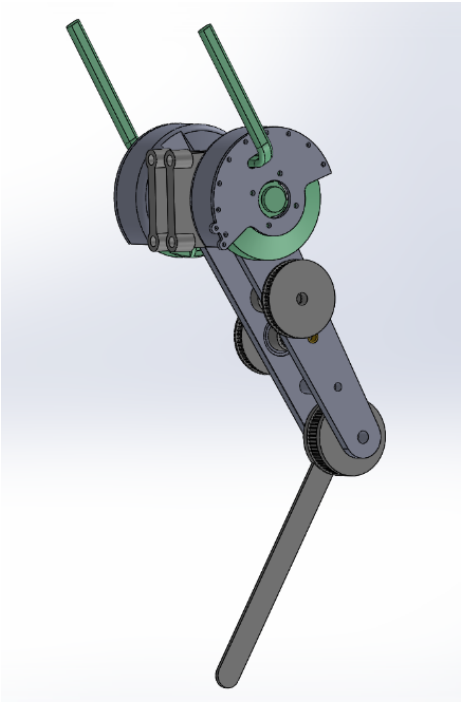


Figure 8: *CAD of First Draft of Extremity*

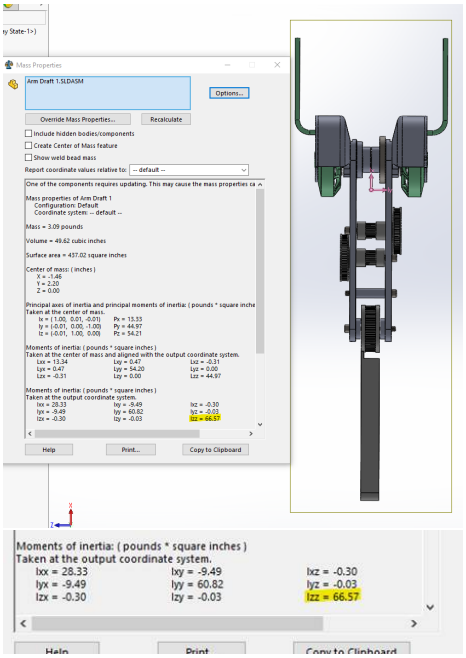


Figure 9: *MOI of First Leg about the Extremity Axle. The total MOI and weight of the first leg was 66.57 lb*in^2 and 3.09 lb respectively*

Figure 10: *Partially Assembled First Version of Leg*

The team learned from the first version of the arm as there are several issues with it. The motors and transmission for both extremities are attached to the upper extremity. This dramatically increases the size and weight of the upper extremity. The motors are not secured well enough in this iteration, causing the tension on the belts to bend the motor mounting and cause the belt to slip. The largest discovery from the first arm had to do with the layout of the joints. The upper extremity joint has two hard end stops, limiting range of motion to less than that required of the front legs. These issues are addressed in the next iteration of the arm.

4.2.1.2 Second Design

Figure 11: *Version 2 CAD*

Figure 12: *Leg Version 2 without electronics or belts*

To solve the range of motion and weight issues with the first arm, the second arm moves the extremity out of plane with the shoulder axis. This adds some complexity controlling the arm as the lower extremity rotates differently, but solves the mass, wiring and range of motion issues. The new layout of the arm dramatically reduces the mass and MOI of both extremities. To prevent the leg from tangling or destroying its own wiring, the encoder and motor wiring are now run internal to the arm. The range of motion of this arm is improved as the end stops are removed. The extremity can rotate continuously about its shaft without limitation.

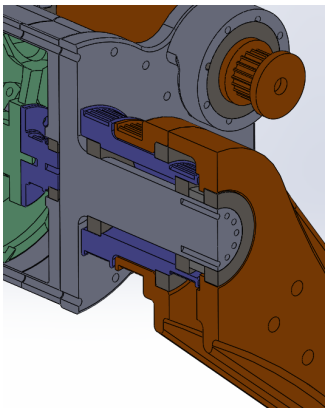


Figure 13: *Cross Sectional View of the interior of the arm*

Mounting the arm on the side of the shoulder housing adds some benefit and novelty to Swol Kat. The extremity can now rotate continuously unrestricted. Although this is not necessary, this could be used to perform behaviors other quadrupeds would not be capable of. For typical robot arms, the angle of each link to the ground is dependent on the angle of the link preceding it. A novelty of Swol Kat's transmission is the angle to the ground of the upper and lower extremities are completely independent of one another.

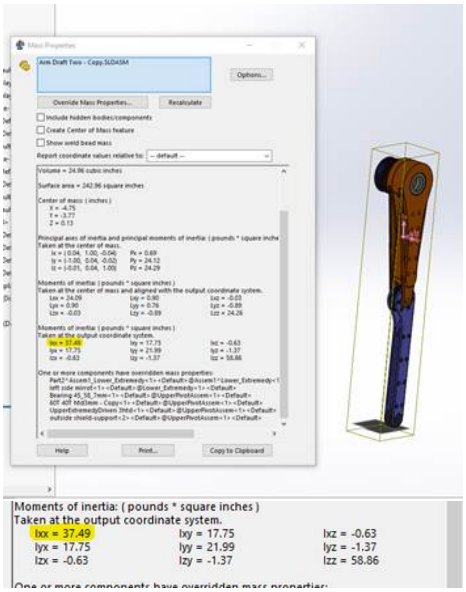


Figure 14: *Extremity MOI about axle*

The MOI of the total extremity is 37.49 lb*in^2 and has a weight of 0.95 lb.

This arm has a moment of inertia that is 70% less than the previous version. This shows that moving the transmission and motors from the arm dramatically reduces the MOI and mass of the extremity.

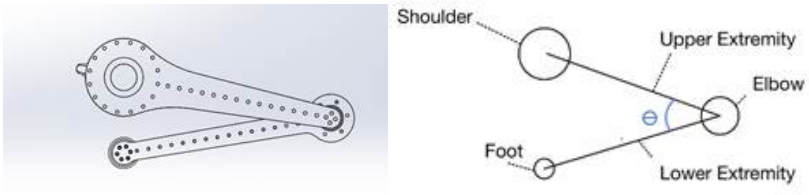


Figure 15: *CAD and Diagram of Extremity*

In the diagram above the leg is in a crouching position. A free body diagram (FBD) was made to solve for the joint torques in this position. Based on the joint torques in this scenario, the gear ratios for the upper and lower extremities were determined. Although impossible, this analysis was performed on largest static loading condition where both the links are parallel to the ground.

4.2.1.3 Third Design

Other successful quadruped projects[4] can independently lift up to 1.6 times the weight of the total robot from the crouching position. The team set a goal that the designed leg would be able to lift 1x the target weight of the robot from the crouching position while not exceeding the motor current rating.

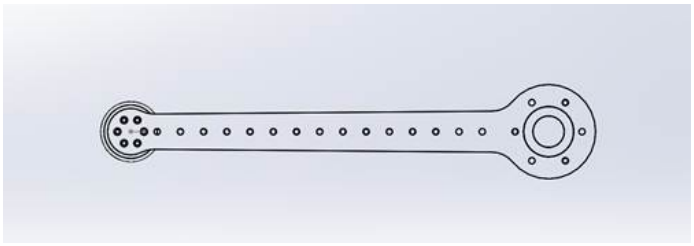


Figure 16: *Lower Extremity*

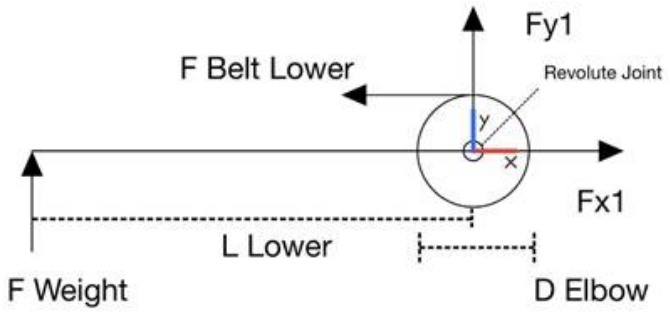


Figure 17: *FBD of Lower Extremity*

All the forces acting on the lower extremity are put onto the FBD and the knowns and unknowns are identified. The unknowns are then solved for. One of the unknown in the FBD is the tension in the belt. The elbow pulley diameter, the mass estimate of Swol kat and length of the lower extremity are known.

$$L_{Lower} = 200 \text{ mm}$$

$$Mass_{SwolKat} = 10 \text{ kg}$$

$$D_{Elbow} = 44 \text{ mm}$$

Table 1: *Known Constants for FBD in Figure: 17*

$$\sum F_y = 0 \qquad \sum F_x = 0$$
$$0 = -F_{y1} - F_{weight} \qquad 0 = -F_{belt} + F_{x1}$$
$$F_{y1} = -F_{weight} \qquad F_{x1} = F_{belt}$$
$$F_{y1} = -M_{quadruped} * g \qquad F_{x1} = 890N$$
$$F_{y1} = -10kg * 9.8\frac{m}{s^2}$$

$$F_{y1} = -98N \qquad F_{x1} = 890N$$

$$
\begin{aligned}
\sum M_z &= 0 \\
0 &= -F_{weight} * L_{Lower} + F_{belt} * \frac{D_{elbow}}{2} \\
F_{belt} &= \frac{F_{weight} * L_{Lower}}{D_{elbow}/2} \\
F_{belt} &= \frac{M_{quadruped} * g * L_{Lower}}{D_{elbow}/2} \\
F_{belt} &= \frac{10kg * 9.8\frac{m}{s^2} * 200mm}{44mm/2} \\
F_{belt} &= 890N
\end{aligned}
$$

Figure 18: *Equations of Equilibrium for Lower Extremity*

Using the tension in the belt and the known diameter of the elbow pulley, the torque applied by the pulley is determined. Based on this torque and the desired maximum torque of the motor the transmission speed ratio is found.

$$
\begin{aligned}
Speed\ Ratio &= \frac{T_{Lower\ Extremity}}{T_{Motor}} * \frac{1}{efficency^{N_{stages}}} \\
&= \frac{19.6Nm}{3.3Nm} * \frac{1}{0.9^3} \\
&= 8.15 \\
&\approx 9
\end{aligned}
$$

Figure 19: *Torque Reduction Calculations for Lower Extremity*

The reduction for the lower extremity is solved for and the limb is determined to need a minimum of an 8.15:1 speed ratio. This ratio was rounded up to 9:1 for a whole number. Next, the same process was applied to the upper extremity to find its gear ratio.

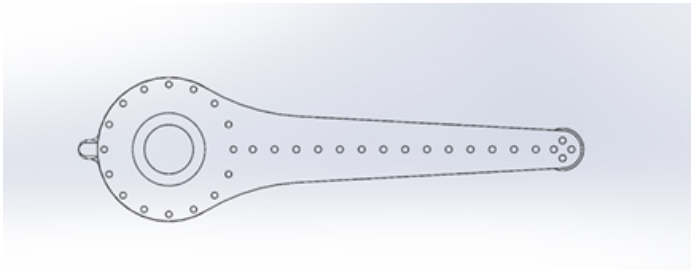


Figure 20: *Upper Extremity*

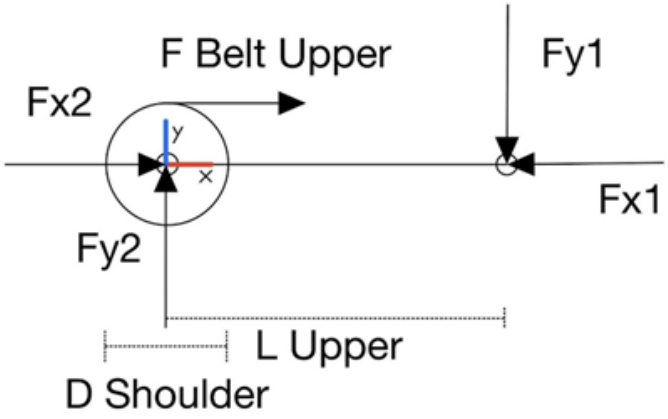


Figure 21: *FBD of Upper Extremity*

All the forces acting on the upper extremity are put onto the FBD and the knowns and unknowns are identified. The shoulder pulley diameter and length of the upper extremity are known prior to calculations. These equations are solved for the torque applied by the pulley.

$$L_{Upper} = 200 \text{ mm}$$

$$D_{Shoulder} = 56.5 \text{ mm}$$

Table 2: *Known Constants for FBD in Figure: 21*

$$\begin{aligned} \sum F_y &= 0 & \sum F_x &= 0 \\ 0 &= F_{y2} - F_{Fy1} & 0 &= F_{x2} - F_{x1} + F_{belt\ upper} \\ F_{y2} &= F_{y1} & F_{x2} &= F_{x1} - F_{belt\ upper} \\ F_{y2} &= -98N & F_{x2} &= 890N - 694N \\ & & F_{x2} &= 197N \end{aligned}$$

$$F_{y2} = -98N \qquad F_{x2} = 197N$$

$$
\begin{aligned}
\sum M_z &= 0 \\
0 &= -F_{y1} * L_{Upper} - F_{belt\ upper} * \frac{D_{elbow}}{2} \\
F_{belt\ upper} * \frac{D_{elbow}}{2} &= -F_{y1} * L_{Upper} \\
F_{belt\ upper} &= \frac{-F_{y1} * L_{Upper}}{D_{elbow}/2} \\
F_{belt\ upper} &= \frac{-98N * 200mm}{56.5mm/2} \\
F_{belt\ upper} &= 694N
\end{aligned}
$$

Figure 22: *Equations of Equilibrium for Upper Extremity*

Using this torque and the desired maximum torque of the motor the speed reduction is found.

$$
T_{Upper\ Extermity} = F_{belt\ upper} * \frac{D_{elbow}}{2} \qquad = 694N * \frac{.0565m}{2} = 19.6\ Nm
$$

Figure 23: *Torque Calculation of Elbow Pulley*

$$
\begin{aligned}
SpeedRatio &= \frac{T_{Upper\ Extremity}}{T_{Motor}} * \frac{1}{efficency^{N_{stages}}} \\
&= \frac{19.6Nm}{3.3Nm} * \frac{1}{0.9^2} \\
&= 7.3 \\
&\approx 9
\end{aligned}
$$

Figure 24: *Torque Reduction Calculations for Lower Extremity*

This shows that the minimum speed ratio of the upper extremity transmission is 7.3:1. This ratio was rounded up to 9:1 so that transmissions for the upper and lower extremities are the same ratio.

Using the results from the torque reduction calculations, the upper and lower extremity were designed. Either side of the extremity sections are made of carbon fiber which help maintain center distances of the upper extremity belt. Carbon fiber was used as it has a similar strength to aluminum but lower density, resulting in a lighter assembly.

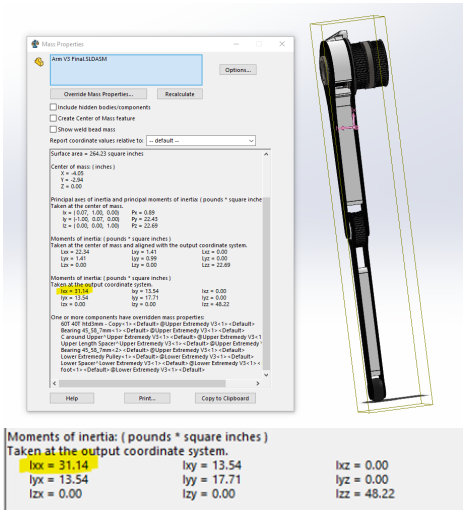


Figure 25: *Extremity MOI about axle. The MOI of the total extremity was 31.14 lb*in^2 and had a weight of 1.02 lb*

4.2.2 Shoulder Joint Design

The shoulder joint of Swol Kat is challenging because a large reduction with low backlash must fit into a small space. During the single arm prototyping, the arms used a belted transmission that was not a high enough reduction nor fit in the required space. This belt slipped very easily under load so it could not be used on the final robot.. Rather than attempting to add tensioners and use a belt, a different approach was taken.

There have been several novel approaches to fitting reductions into quadrupeds that are highly efficient, light and compact. One caught the team's attention because of its unique abilities: the capstan. The capstan operates similarly to a gear in the sense that two circular parts roll against one another producing a change in torque from one to the other. However, the tooth interface between gears puts the blunt of the force on a small surface area. This is not friendly for 3d printing as it will wear the teeth quickly. Capstans distribute the forces at the interface over a much larger area allowing them to be 3D printed.

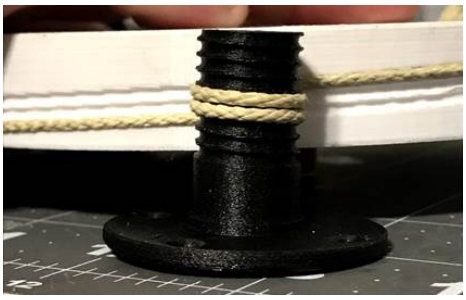


Figure 26: *Prototype 1 of Swol Kat Capstan Shoulder*

Capstans consists of a rope wrapped multiple times about a drum. The friction between the drum and the rope allows the drum to be rotated applying a tensile load to one end of the rope while giving slack to the other. By attaching either end of the rope to a driven cylinder the drum can essentially roll against the driven cylinder allowing for a system to have minimal backlash and transmit a large amount of torque. An added benefit to capstans is the two rollers draw themselves together, unlike gears which actively push themselves apart.

4.2.2.1 Capstan Prototype

Unlike belts, which come in set sizes, the capstan is fully 3D printed and any rollers can be made to work with any spacing. The ideal spacing of the joint in the robot geometrically is 75mm so the driving and driven rollers have a diameter of 15mm and 135mm respectively. This resulted in a 9:1 speed reduction.

$$F_{max\ tension\ in\ string} = \frac{T_{motor\ stall}}{\frac{D_{driving\ roller}}{2}}$$

$$F_{max\ tension\ in\ string} = \frac{3.92Nm}{\frac{0.15m}{2}}$$

$$F_{max\ tension\ in\ string} = 523N$$

Figure 27: *String Tensile Strength Requirement*

The string material chosen for the capstan was Dyneema. This is because of its high tensile strength to weight ratio. Dyneema fiber has 15 times the tensile strength of steel by weight[15]. To obtain a high factor of safety and increase the ease of assembly, Dyneema that is 1/16” diameter is used even though its tensile strength is far above the requirements.

$$F_{dyneema\ yield\ tension} = 450lbf \approx 2001N$$
$$Factor\ of\ Safety = \frac{f_{dyneema\ yield\ tension}}{f_{max\ tension\ in\ string}}$$
$$Factor\ of\ Safety = \frac{2001N}{523N}$$
$$Factor\ of\ Safety = 3.826$$
$$Factor\ of\ Safety \approx 4$$

Figure 28: *Factor of Safety(FOS) on Dyneema*

4.2.2.2 Determining the wraps necessary for the minor roller

To determine the number of wraps necessary for the minor roller the coefficient of static friction between the driving roller and the rope must be known. A test was performed to determine the coefficient of static friction. The driving roller is printed out of NylonX[16], so a sample was printed to act as a test surface. A weight was calibrated to a mass of 1 kg. A loop of the Dyneema was placed between the calibrated weight and the test surface and a force gauge pulled horizontally on the Dyneema[17].

Figure 29: *Calibration of Weight*

Figure 30: *Full Dyneema NylonX Friction Test Setup*

Figure 31: *Calibrated Weight, Dyneema, NylonX Sample Stack*

Figure 32: *Force Gauge Reading when the Static Friction Force Was Broken*

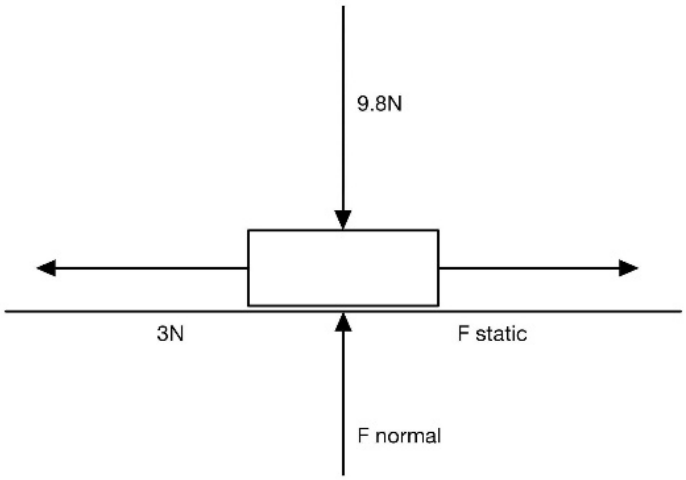


Figure 33: *FBD of Dyneema under Weight on NylonX*

In the FBD above the Dyneema and weight are treated as a single object. Friction between the Dyneema and weight is unopposed, so they do not move relative to one another.

$$\begin{aligned}
\sum F_y &= 0 \\
0 &= F_{normal} - 9.8N \\
F_{normal} &= 9.8N \\
\sum F_x &= 0 \\
0 &= -3N + F_{static} \\
F_{static} &= 3N \\
\mu_s * F_{normal} &= 3N \\
\mu_s * 9.8N &= 3N \\
\mu_s &= 3N/9.8N \\
\mu_s &= .41
\end{aligned}$$

Figure 34: *Solution For Coefficient of Static Friction*

The force applied by the spring scale at the point the Dyneema begins to slip against the NylonX is approximately 3 Newtons. Using this information with the known weight, and assuming the Dyneema's mass is negligible, the coefficient of static friction is calculated to be approximately 0.3.

The formula for the holding force of a capstan[1] is show in equation **??**. Where T_{load} is the loaded side tension. The three variables that change the holding force of the capstan are: the slack side tension(T_{hold}), the coefficient of static friction between the roller and rope(μ), and the angle of wrap around the capstan roller(ϕ).

$$T_{Load} = T_{hold}(e^{\mu * \phi})$$

Figure 35: *The Capstan Equation[1]*

$$\begin{aligned}
T_{load} &= T_{hold} * e^{\mu\phi} \\
T_{hold} * e^{\mu\phi} &= T_{load} \\
e^{\mu\phi} &= \tfrac{T_{load}}{T_{hold}} \\
ln(e^{\mu\phi}) &= ln(\tfrac{T_{load}}{T_{hold}}) \\
\phi &= ln(\tfrac{T_{load}}{T_{hold}})/\mu \\
N_{wraps} &= \tfrac{\phi}{2\pi} \\
N_{wraps} &= \tfrac{ln(\frac{T_{load}}{T_{hold}})/\mu}{2\pi}
\end{aligned}$$

Figure 36: *Solving the Capstan Equation for Wrap*

A benefit of the capstan over timing belts is the ability to get multiple full wraps of the driving roller. To determine the number of wraps necessary the equation in Fig X can be rewritten to solve for radians of wrap given the other parameters. The solution is then rounded to the next highest rotation. For the first prototype the assumption was made that 3 lb of pretension could be held on the slack side of the capstan.

$$\begin{aligned}
N_{Wraps} &= \tfrac{\ln(T_{load} \ / \ T_{hold})}{4*2*\pi} \\
N_{Wraps} &= \tfrac{\ln(523N \ / \ 3lbf)}{.3*2\pi} \\
&= 1.95 \\
&\approx 2
\end{aligned}$$

Figure 37: *Number of Wraps Necessary Given Pretension of 3 lb-f*

In the first capstan prototype the driving and driven rollers has a diameter of 15mm and 135mm respectively. Either end of the capstan has the Dyneema tied down to a screw threaded into the driven roller. This iteration has two wraps of Dyneema around the driving roller.

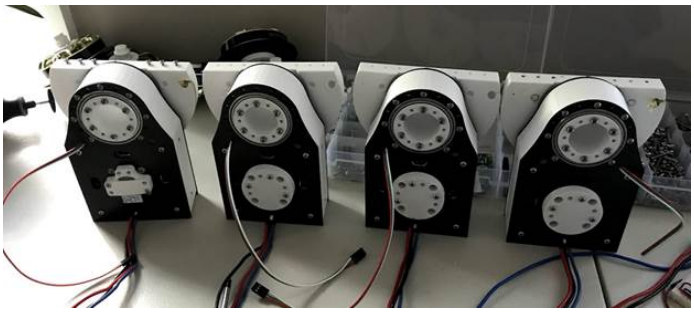


Figure 38: *First Capstan Prototypes*

Proper pretension could not be kept on this iteration and the Dyneema slipped on the driving roller. An additional 2 wraps were added to the driving roller, bringing the total to four wraps. This brought the pretension requirements down to approximately 1 oz-f. To increase ease of assembly a tensioning mechanism was added.

$$
\begin{aligned}
N_{Wraps} &= \frac{\ln(T_{load} \ / \ T_{hold})}{4 * 2 * \pi} \\
N_{Wraps} &= \frac{\ln(523N \ / \ 0.25lbf)}{.3 * 2\pi} \\
&= 3.26 \\
&\approx 4
\end{aligned}
$$

Figure 39: *Number of Wraps Necessary Given Pretension of 0.25 lb-f*

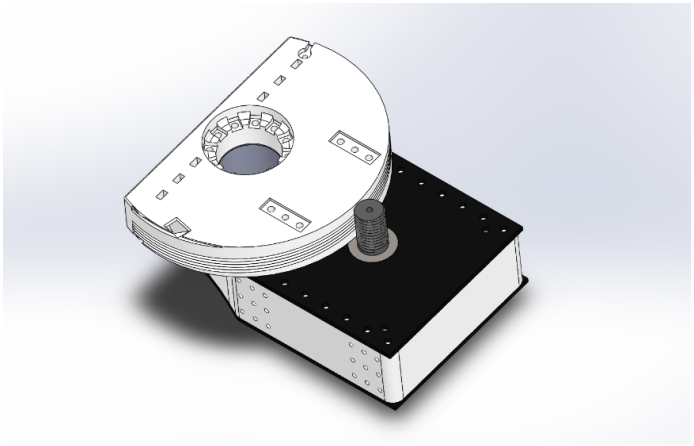


Figure 40: *CAD of Final Capstan Prototype*

Figure 41: *Final Capstan Prototype*

In the final capstan prototype the Dyneema did not slip against the driving roller at any point during testing.

4.3 Controls

For control of the leg a joint tip position control operates alongside a joint tip force limiting algorithm.

4.3.1 Forward Kinematics

The forward kinematics are used to convert from robot joint angles (θ_1,θ_2,θ_3) to robot task space (x,y,z). To calculate the forward kinematics the David Hardenburgh [18] convention is used. The first step is assigning frames to the robot arm in its zero configuration. Due to the orientation of the arms on the quadruped (specifically where the zero positions of the legs are) the forward kinematics are solved for two different zero positions.

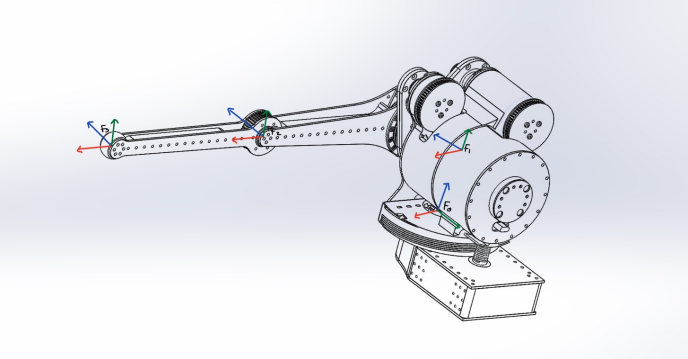


Figure 42: *Arm Home in First Configuration*

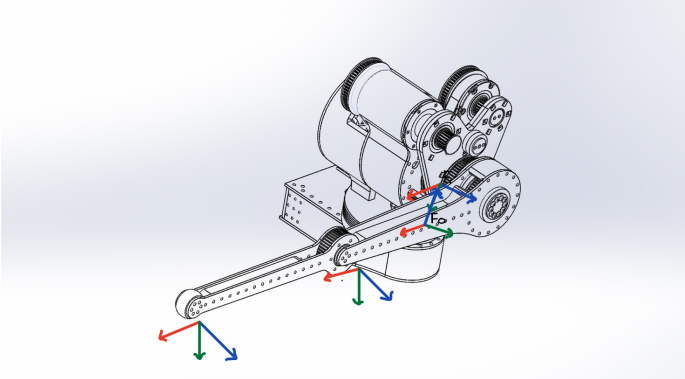


Figure 43: *Arm Home in Second Configuration*

Using these frames the DH table for each solution can be calculated.

	θ	d	a	α
T_0^1	θ_1	D1	0	$-\pi/2$
T_1^2	0	D2	A2	0
T_2^3	$\theta_3 - \theta_2$	0	A3	0

Table 3: *DH table for first configuration*

	θ	d	a	α
T_0^1	θ_1	D1	0	$\pi/2$
T_1^2	0	D2	A2	0
T_2^3	$\theta_3 - \theta_2$	0	A3	0

Table 4: *DH table for second configuration*

Each row of the DH table is then converted into a 4x4 transformation using the homogeneous transform matrix (1).

$$T_{homogeneous}(\theta,d,a,\alpha) = \begin{bmatrix} \cos(\theta) & -\sin(\theta)*\cos(\alpha) & \sin(\theta)*\sin(\alpha) & a*\cos(\theta) \\ \sin(\theta) & \cos(\theta)*\cos(\alpha) & -\cos(\theta)*\sin(\alpha) & a*\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Multiplying each matrix together gives a transformation matrix that can convert from joint space to task space.

$$T_0^3 = T_0^1 * T_1^2 * T_2^3$$

Plugging in our joint angles to T_0^3 will give the orientation and position of the end effector

4.3.2 Inverse Kinematics

Inverse Kinematics are used to convert from task space (x,y,z) to joint space ($\theta_1, \theta_2, \theta_3$). There two typical to ways to calculate these equations: the algebraic method and the geometric method. The team decided to go with the geometric method as the configuration of the leg is similar to the shoulder offset manipulator. [19] While the shoulder offset leg has up to 4 possible inverse kinematic solutions as shown in picture (x) due to the configuration of the legs on the robot each of the zero configurations only need to be solved for two of these solutions.

$$r = \sqrt{x^2 + y^2}$$
$$\alpha = \text{atan}_2(y,\ x)$$
$$u = -\sqrt{r^2 - D2^2}$$
$$\beta = \text{atan}_2(D2,\ u)$$

$$\theta_1 = \alpha - \beta$$

$$s = z - D1$$
$$D = \frac{r^2 2 + s^2 - A2^2 - A3^2}{2 * A2 * A3}$$
$$\phi = \text{atan}_2(s,\ r)$$
$$\gamma = \text{atan}_2(a3 * \sin t3,\ a2 + a3 * \cos t3)$$

$$\theta_2 = -(\phi + \gamma)$$
$$\theta_3 = \theta_2 - \text{atan}_2(\pm\sqrt{1 - D^2},\ D)$$

$$r = \sqrt{x^2 + y^2}$$
$$\alpha = \text{atan}_2(y,\ x)$$
$$u = \sqrt{r^2 - D2^2}$$
$$\beta = \text{atan}_2(D2,\ u)$$

$$\theta_1 = \alpha + \beta$$

$$s = z - D1$$
$$D = \frac{r^2 2 + s^2 - A2^2 - a3^2}{2 * A2 * A3}$$
$$\phi = \text{atan}_2(s,\ r)$$
$$\gamma = \text{atan}_2(a3 * \sin t3,\ a2 + a3 * \cos t3)$$

$$\theta_2 = \phi + \gamma$$
$$\theta_3 = \theta_2 + \text{atan}_2(\pm\sqrt{1 - D^2},\ D)$$

4.3.3 Jacobian

The Jacobian is 6xn matrix used to translate between joint space velocities and task space velocities.

$$\dot{p} = J(q) * \dot{q}$$
$$\dot{q} = J(q)^{-1} * \dot{p}$$

The Jacobian is also capable of translating between joint space torques and task space forces. Solving the Jacobian allows us to do force control on the tip of the leg.

$$\tau = J(q)^T * F_{tip}$$
$$F_{tip} = (J(q)^T)^{-1} * \tau$$

There two methods solving the Jacobian: the derivative method and the cross product method. The cross product method was selected as it uses the transformation matrixes calculated in the forward kinematics. This allows for a single method for solving the Jacobian that is not dependent on the zero configuration

of the leg.

$$J = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$$

J_p and J_o represent the position and orientation halves of the Jacobian. Each one is made up of a i columns where i corresponds to the i^{th} joint on the robot. J_{pi} and J_{oi} can be calculated using the formulas below where z_i is the z column of the rotation matrix for the i^{th} joint, p_e is the position of the end effector and p_i is the position of the i^{th} joint.

$$J_{pi} = z_i \times (p_e - p_i)$$
$$J_{oi} = z_i$$

4.3.4 Current Limiting

In each loop of the controller, the current limits of the motor can be adjusted to limit the maximum torque that the motor will apply. In order to limit the damage accidental collisions can cause the motors receive a new torque limit to prevent the leg from applying too much force. To find the maximum torque values for the motor the Jacobian, current tip position, and maximum desired tip force of the arm are used in the calculation. To calculate the current limit of each leg motor, first the contribution to the tip force by the torque of the other motors is calculated.

$$\begin{bmatrix} T1 \\ T2 \\ 0 \end{bmatrix} * (J(q)^T)^{-1} = F_{tip\_2motor}$$

Next, this force value is subtracted from the tip force limit, to find how much more force can be applied by the tip without exceeding it. This remaining tip force is then converted back into joint torques

$$F_{remaining\_min} = -1 * F_{max} + F_{tip\_2motor}$$

$$F_{remaining\_max} = F_{max} + F_{tip\_2motor}$$

$$F_{remaining} * J(q)^t = \begin{bmatrix} T1 \\ T2 \\ T3 \end{bmatrix}$$

The resulting joint torque is then the minimum and maximum torque limits for the motor. Due to how the ODrive motor controllers handle torque limiting, only an absolute torque limit can be set. The torque limit that is sent to the ODrive motor controller is shown below. Using this torque limiting algorithm allows the maximum allowable tip force on the legs to be reduced, minimizing the damage an accidental collision can cause.

$$Min(Abs(T_{min}), Abs(T_{max}))$$

4.3.5 Quintic Trajectory Planning

In order to test the forward and inverse kinematics of the single leg the team implemented quintic trajectory planning allowing the arm to plan a path from one point to another. Quintic trajectory planning is useful as it allows control over end effector position, velocity, and acceleration. Traditionally the coefficients for a the quintic polynomial function can be solved using the following matrix formula.

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ a_0 \\ q_f \\ v_f \\ a_f \end{bmatrix}$$

However, while controlling the arm the formula is only solved for zero start and end velocities and zero start and end accelerations. To save computation time and simplify the code the quintic polynomial is pre-solved for a 1 second movement from 0-1 as shown below.

$$p = 0 + 0t + 0t^2 + 10t^3 - 15 * t^4 + 6 * t^5$$

p is then solved for by plugging in the percentage of completed motion as t, p was then scaled to an x, y, z position by using the equation below where p_t was next target position of the robot, p_f is the goal position, and p_i was start position.

$$p_t = (p_f - p_i) * p + p_i$$

4.4 Software Stack

The software stack for the arm was designed to be directly usable in the final control system. The software stack for a single arm was designed to be usable in the test environment and the final robot.

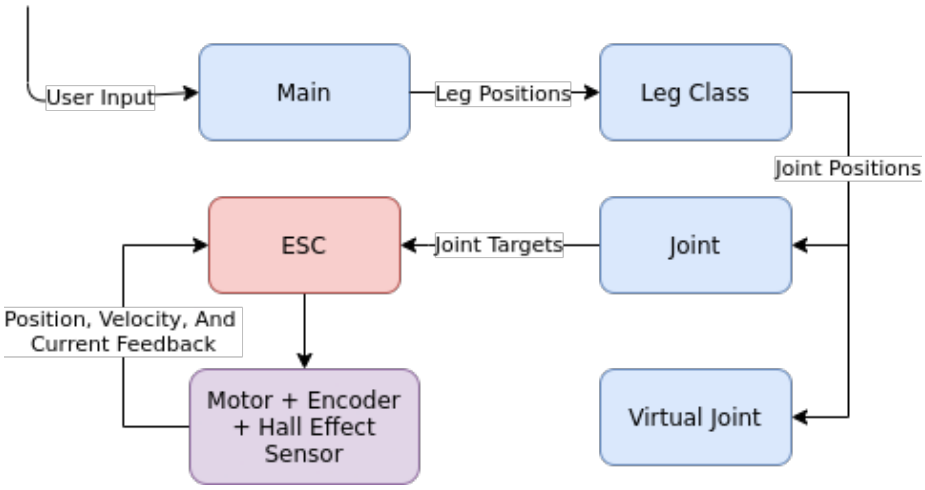


Figure 44: *Single Arm Overview*

4.4.1 ODrive Tuner

At the start of the project, working with the ODrive command line tool proved rather difficult, especially when it came to tuning the controller. For this reason a web based interface was developed 69. The tool was based on Socket.io[20] and Flask[21] which allows real time communication with the computer, and has a variety of useful features:

- Connecting to, Rebooting and Clearing Errors of the ODrive.
- Live Reporting on any errors that occurred for the axis that was being worked on.
- Live Readout of Encoder Position, Voltage, and Current State.
- Ability to set a variety of Axis States and Control Modes.
- Seeing and Setting Position, Velocity and Current setpoints.
- Seeing and Setting Gains and Config of the ODrive.
- Live Graphing of Position, Velocity, and Current setpoints and positions.
- A debugging log to display useful information.

4.4.2 Joint and Virtual Joint Class

The lowest level class developed were Joint and Virtual Joint. These classes are responsible for managing all joint level information. They are responsible for both communication with the ODrive as well as storing the position, velocity, and

torque data of each joint. The Joint class communicates with the ODrive to get this information and takes in a ODrive Axis object during creation. The Virtual Joint emulates communication with the ODrive and can be used in place of the Joint class. This modular design allows for easy switching between a physical and virtual testbed. Additionally, the physical and virtual components can be mixed allowing for testing portions of the physical robot.

4.4.3 Leg Class

The Leg class controls 3 individual joints to make them function as a single leg. This class is constructed with the 3 Joint (or Virtual Joint) objects, the link lengths of the leg, and the corner of the robot that the leg belongs to. The legs are configurable via a single JSON file.

4.4.4 Leg Stick Plot

A visualization of the leg was developed to test the inverse kinematics, forward kinematics, and force sensing on the tip. The visualization is based on matplotlib [22] and continuously plots the arms current angles using the forward kinematics.

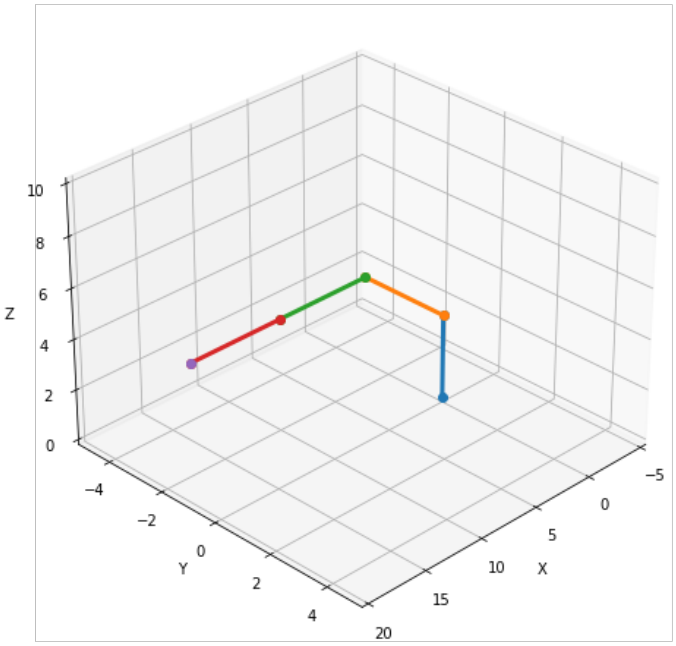


Figure 45: *Stick Plot Of Leg*

4.4.5 Leg UI

In order to test a leg a web based UI for leg control was developed 70. This UI was also based on Socket.io [20] and a Flask Server [21]. This system offloaded rendering of the leg to the client, freeing up resources on the central controller. The Leg UI also has a variety of other features:

- Communicate wirelessly over a LAN between the Raspberry Pi and a users computer using a web browser
- Visualize the current position of each limb of the robot leg
- Calibrate, Home, and Enable the robot leg
- Individually jog each joint or the tip of the robot in x,y,z
- Display Current, Status, and Errors of the System.

Include Picture of the GUI may also need to go into appendix.

4.5 Communication

The distributed design requires development of communication links between controllers. The first link that was designed was between the ODrive and Teensy. This link needed to be high bandwidth (1khz) to ensure the link did not throttle the control system. A control library and protocol already existed to communicate with the ODrive over the UART interface. When testing the ODriveArduino C++ library, which uses ODrive's ASCII Protocol, there were many shortcomings such as a lack of flexibility in the types of control messages supported, and a lack of feedback. This resulted in rewriting the majority of the library to add support for the ODrive functionality we required. The rewritten library is designed to allow for more independent control of each ODrive channel, and allows for the Teensy to simultaneously control 3 ODrives.

5 Design/Development of Quadruped

5.1 System overview

Figure 46: *Render of Final Quadruped*

The final quadruped is the result of combining the independent components previously developed. This required fully implementing the communication and control system, as well as incorporating all the systems components into a neat form factor.

5.2 Mechanical

5.3 Communication

In order to control all 4 limbs simultaneously the Teensy to Raspberry Pi communication link had to be developed. When outlining the requirements of this link, the role of the Teensy was to receive tip position setpoints from the Raspberry Pi, perform inverse kinematics and send joint position setpoints to the ODrives. For testing purposes, an additional requirement was that the Teensy could also receive motor positions directly from the Raspberry Pi and communicate them to the ODrives.

After looking at the additional overhead required to build the link with individual control for each motor, it was determined that the best solution was to bypass the Teensy and connect the ODrives directly to the Raspberry Pi. This decision also took into account the faster development speed of python, rather than splitting development between python and C++.

The ODrive Python SDK that the new communication system is built upon can use a higher default baudrate, along with smaller packets. It was also determined that the ODrive and motor respond to position setpoints with a cutoff frequency of 17hz, reducing the requirements of the communication link. Removing the Teensy comes at the cost of increasing the processing done by the Raspberry Pi, limiting the remaining resources for future development. There were plenty of resources for the current project goals.

When connecting the ODrives to the Raspberry Pi, we started by using a single USB hub that could connect to all 6 ODrives. This caused random dropouts, which were determined to be caused by overloading the USB Bus. This was solved by using two USB hubs in the robot, one for each half of the robot, and connecting them to USB ports that were on separate USB controllers.

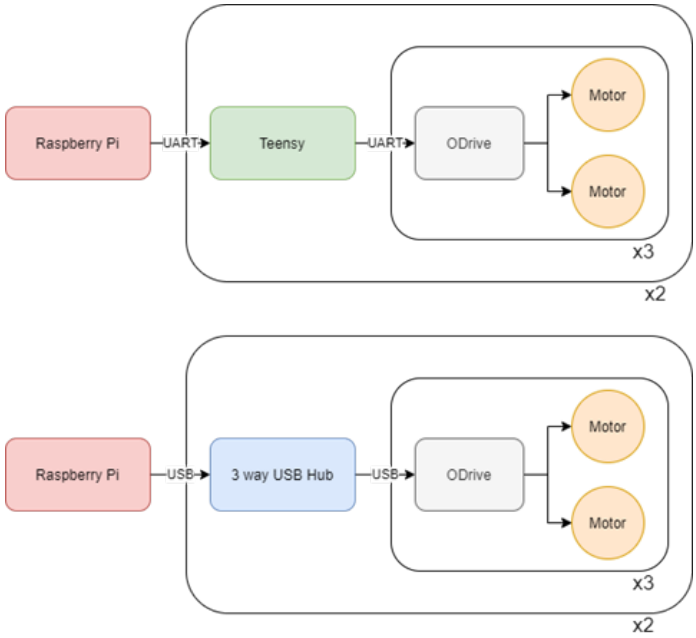


Figure 47: *Initial control system diagram (top) and final control system diagram (bottom)*

5.3.1 Multi Threading

Due to how the ODrive communication library is structured, each time a value is changed on an ODrive a blocking function must be called. If each ODrive is sequentially written to in a single thread, the loop speed of the program sits at only 8hz. To speed up the communication process, a thread was created for each ODrive that handles communication with the single controller. This allows for the blocking functions of each ODrive to be handled separately from the main loop. By separating the ODrive communication functions from the main loop, the loop

speed was increased from 8hz to about 45hz. Running the main loop at 45hz is more than double the position control bandwidth of the motors so there is almost no performance loss due to the main loop speed.

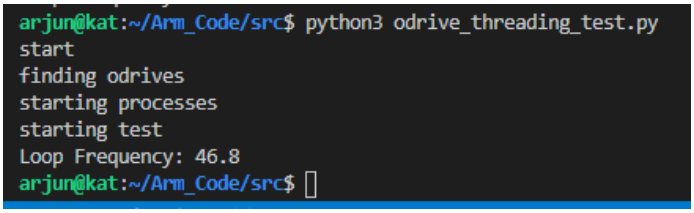


Figure 48: *Results from a program that runs the main loop for 5 seconds to determine the loop frequency.*

To relay information from the main thread to the ODrive threads, an inter-thread communication tool called a pipe was used. When a pipe is created, two endpoints are made. One endpoint is passed to the ODrive thread, and the other is kept by the main thread. With these endpoints, data that can be packaged into a byte stream can be sent between the threads to relay information. On every iteration of the main loop, a data structure is sent to the ODrive thread that contains the position setpoint and current limit for the motor. Optionally, a pointer to a function can be sent to the ODrive thread for it to run on the ODrive object. This allows for more advanced functionality like homing, error checking, and calibration to be run on the thread without having to build a complex state machine on the ODrive thread side.

The main loop then pauses after the information is sent to the ODrive threads and waits until a response is received. This response to the main thread tells the main thread that the values have been written to the ODrives and relays any additional information that may be needed. For example, when the ODrive is polled for errors a packet is returned to the main thread that tells it what errors exist, if any.

5.3.2 ODrive Firmware

ODrive claims to natively support the AS5047P encoder[11], so the robot was built using that encoder on all of the motors. The ODrive communicates with the encoders well in low noise environments, like when using a single motor on a test bench, but when a single bit error can be triggered on the MOSI line, the ODrive encoder communication breaks down.

The AS5047P encoder has a feature that when receiving data over SPI, a bit in the received packet tells the bus master whether an invalid command has been sent to the encoder.[2] The master must issue a special command packet that clears the error bit. The software running on the ODrive motor controller checks this bit to see if a received packet has good data but does not send the reset command if the bit is set. As a result, if there is enough noise on the MOSI line to cause a single bit error, the error flag will be set and never reset, causing

the ODrive to invalidate all of the incoming data. After a short period of not receiving good data, the ODrive will throw an encoder error and halt operation

SPI Read Data Frame

Bit	Name	Description
15	PARD	Parity bit (even) calculated on the lower 15 bits
14	EF	0: No command frame error occurred 1: Error occurred
13:0	DATA	Data

The parity bit PARD is calculated by the AS5047D of the lower 15 bits of data frame. If an error occurred in the previous SPI command frame, the EF bit is set high. The SPI read is sampled on the rising edge of CSn and the data is transmitted on MISO with the next read command, as shown in Figure 15.

Figure 49: *SPI read data frame and EF bit description from the AS5047P encoder datasheet.[2]*

With the ODrive unable to send the encoder the correct error reset packet, the only way to reset the system to a functional state is to restart the encoder. The error is saved in volatile memory so it is cleared when the encoder restarts.

During single motor testing this error was rare because the motor wires were further from the encoder signal wires and only one encoder was connected to the ODrive. The motor wires effectively have a three-phase alternating current running through them at the frequency of rotation of the motor with an amplitude of the motor phase current. The EMF produced from these wires are likely a large source of interference to the encoder communications. Additionally, due to how each ODrive controls 2 motors adding a second encoder doubles the length of wire that could be susceptible to interference. Moving to a full arm where the motor wires could not be separated from the encoder wires and two encoders were connected to each ODrive, the encoder errors became far more frequent and made the system nearly unusable.

To fix this issue, the ODrive firmware had to be changed to reset the AS5047P encoder error flag correctly when it is set. The interrupt service routine where encoder data is read was modified to also set a flag when the EF bit is set and the encoder data is ignored. Then, in the section where the encoder read is triggered, the flag is checked to see if an error reset packet should be sent instead of the read encoder position packet.

After compiling and installing the new firmware on the ODrive motor controllers, the encoder error was resolved and the ODrive encoder communication became far more robust.

5.4 Controls

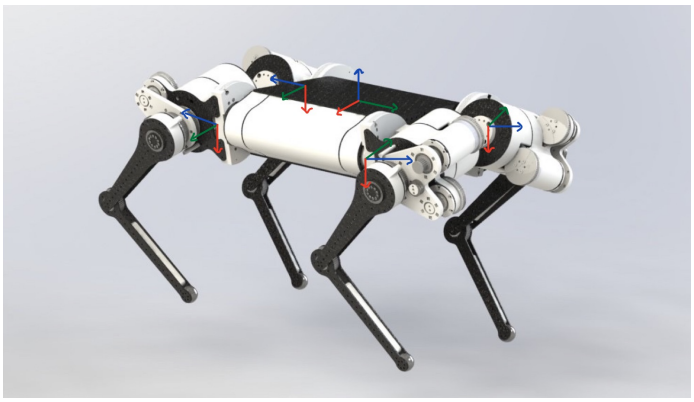


Figure 50: *Leg Frame Locations on Robot*

5.4.1 World To Robot Transformation

The location of the world to robot is defined by a translation(x,y,z) and a rotation(α,β,γ). Using a ZYX rotation matrix (2) a 4x4 transformation matrix can be made that allows the conversion between world coordinates and robot coordinates (3). Where P is a 3x1 matrix that represents the translation of the robot body:

$$R_Z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} R_Y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} R_X(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}$$

$$R_{ZYX}(\alpha,\beta,\gamma) = R_Z(\alpha) * R_Y(\beta) * R_X(\gamma) \tag{2}$$

$$T_W^R(x,y,z,\alpha,\beta,\gamma) = \begin{bmatrix} R_{ZYX}(\alpha,\beta,\gamma) & P(x,y,z) \\ 0 & 1 \end{bmatrix} \tag{3}$$

5.4.2 Robot To Leg Frame Transformation

Each leg has its own transformation from the robot frame to its defined leg frame. A set of euler angles were pre-calculated for each leg in order to define its rotation with respect to the robot frame as seen in Table: 5. The length and width of the robot body are used to define the position of the robot body with respect to the robot frame as seen in Table: 6, where w is the width of the robot and l is the length of the robot. Using the ZYX euler transformation matrix (2) we can make a transformation matrix that converts between the robot frame and body frame.

Leg Number	α	β	γ
1	$\tau/4$	$\tau/4$	0
2	$\tau/4$	$\tau/4$	0
3	$-\tau/4$	$\tau/4$	0
4	$-\tau/4$	$\tau/4$	0

Table 5: *Rotation of Each Leg on Robot*

Leg Number	x	y	z
1	$w/2$	$l/2$	0
2	$-w/2$	$l/2$	0
3	$-w/2$	$-l/2$	0
4	$w/2$	$-l/2$	0

Table 6: *Position of Each Leg on Robot*

5.4.3 Leg Swing Phase Control

A Bezier Curve was generated in order to control the swing phase of the step. The Bezier curve is defined by a series of control points that define the shape of the curve. The formula for a Bezier curve is defined as Equation 4:

$$B(t) = \sum_{i=0}^{n} (\tbinom{n}{i}) P_i (1-t)^{n-i} t^i \tag{4}$$

Where P_i is the fitting point. The Bezier curve also has several other desirable properties: two points at the same position define a zero velocity point and three points at the same position define a zero acceleration point [23]. According to these properties a smooth curve can be obtained using the twelve control points as shown in Table 7. Where s_l is the desired step length and s_h is the desired step height.

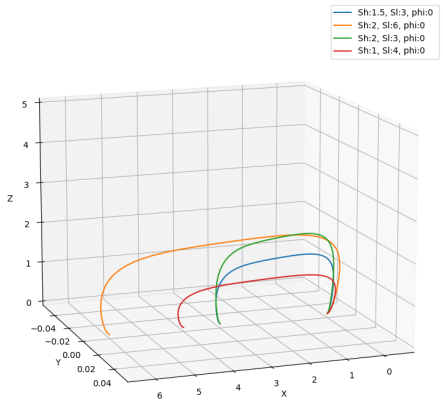


Figure 51: *Bezier curves generated for various step heights and lengths*

P_n	X(mm)	Y(mm)	Z(mm)
P_0	0	0	0
P_1	-$s_l * .05$	0	0
P_2	-$s_l * .1$	0	$s_h * 1.1$
P_3	-$s_l * .1$	0	$s_h * 1.1$
P_4	-$s_l * .1$	0	$s_h * 1.1$
P_5	$s_l * .5$	0	$s_h * 1.1$
P_6	$s_l * .5$	0	$s_h * .9$
P_7	$s_l * .5$	0	$s_h * .9$
P_8	$s_l * 1.1$	0	$s_h * .9$
P_9	$s_l * 1.1$	0	$s_h * .9$
P_{10}	$s_l * 1.05$	0	0
P_{11}	s_l	0	0

Table 7: *Bezier Curve Control Points*

5.4.4 Foot Ground Phase Control

In Ground Phase Control the robot attempts to keep its foot in the same world coordinate as the robot body translates forward. The leg is commanded to go slightly below the ground and the tip force limits prevent the leg pushing too hard, tipping the robot.

5.5 Software Stack

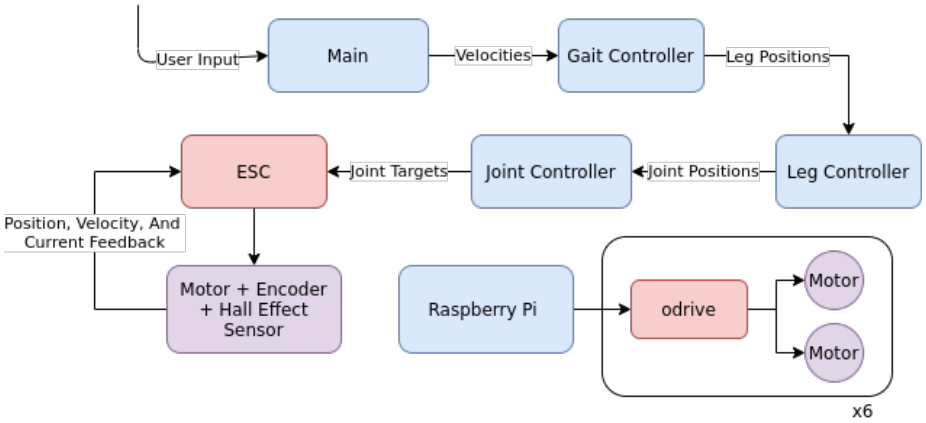


Figure 52: *Full Quadruped Code Overview*

5.5.1 Robot Class

The Robot Class is responsible for managing all robot level information. The class is constructed using 4 leg objects, and loads the robot configuration from a json file. The robot class contains information about the current gait the robot is executing and the current body position of the robot. Given the multithreaded nature of the code the robot object has as single loop function. Loop executes one loop of the current gait and then updates the arm objects.

5.5.2 Gait Class

In order to standardize how gaits are run on the robot a Gait super class was created, which standardizes the structure for each. This structure is outlined in Figure 53.

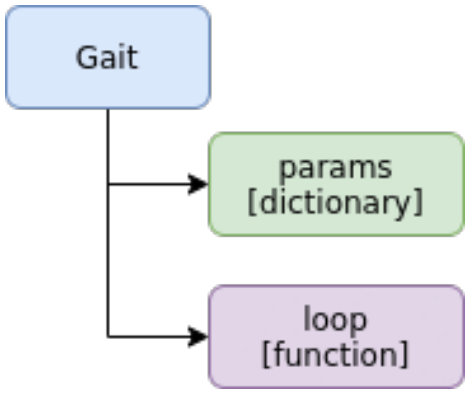


Figure 53: *Basic Structure for all gaits*

This structure creates "hot swappable" gaits. All parameters that control the behavior of the gait exist in the Params dictionary. This provides a common place for a higher level program to look to adjust gait parameters. The loop function executes one loop of the gait. Each loop the gait updates the target positions for each leg of the robot. During execution the robot object is passed into the loop function. Allowing the gait to update the current body position of the robot and access each of the leg objects.

As the code is not a real time system, a constant time per loop can not guaranteed thus each gait class must be able to track and handle variable loop times. This constraint is handled by defining robot movements as continuous functions of time which are evaluated when the loop is executed.

This gait structure allows for the separation of gait selection and management from the gait itself. Each gait is simply responsible for moving the robot given its input parameters.

5.5.3 Virtual Robot

A visualization of the robot was developed to test each of the gaits. The visualization was based on matplotlib [22] and continuously plots the legs and robot current position using the forward kinematics.

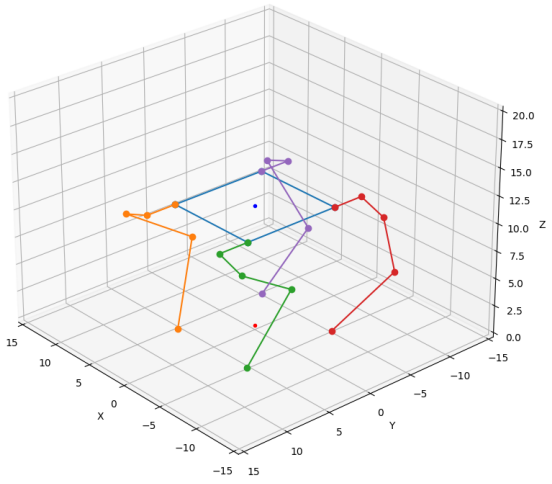


Figure 54: *Plotted Robot*

5.5.4 Utility

Over the course of this project a variety of utility functions or classes were developed to ease the programming and management of the robot.

- BodyState Class: Stores the location and orientation of the robot with respect to the world frame
- Kinematics Module: has a series of helper functions that generate useful transformation matrices like a ZYX Rotation Matrix or the Homogeneous Transform Matrix
- get_body_pts: a function that takes in a BodyState Object and returns where the 4 leg origins would be in world coordinates
- get_leg_rot: a function that takes in a leg number and returns a rotation matrix that corresponds to its rotation from the robot frame.
- Robot Plot and Leg Plot: Classes that can plot the current position of the robot or leg respectively

- Foot Control Module: A Module that takes in percentage of movement completed, step length, height, and rotation and returns a position from a generated Bezier curve that a foot should be at.

5.6 Gaits

5.6.1 Wiggle Gait

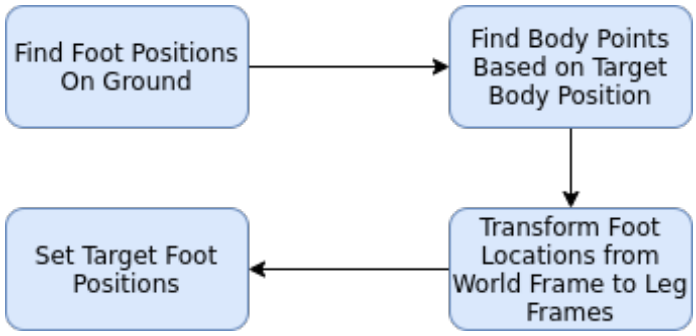


Figure 55: *Logic For Single Loop of Wiggle Gait*

The wiggle gait tries to keep the feet of the robot on the ground while moving the body around. It calculates the foot position using the get_body_pts function so that the robot body is at (0,0,0).This position is then translated from the world frame to individual leg frames. With an appropriate input this gait is able to move the body with 6 degrees of freedom while keeping the feet of the robot planted.

5.6.2 Crawl Gait

The crawl gait tries to move the robot body forward at a constant velocity while moving the legs forward 1 at a time.

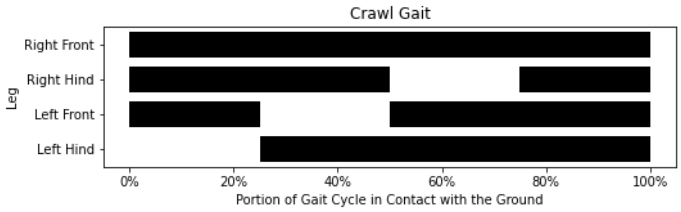


Figure 56: *Step Pattern for Crawl Gait*

The state in Figure 57 is the leg that is preforming the step each cycle, once the step is completed the state is updated to the next leg in the sequence

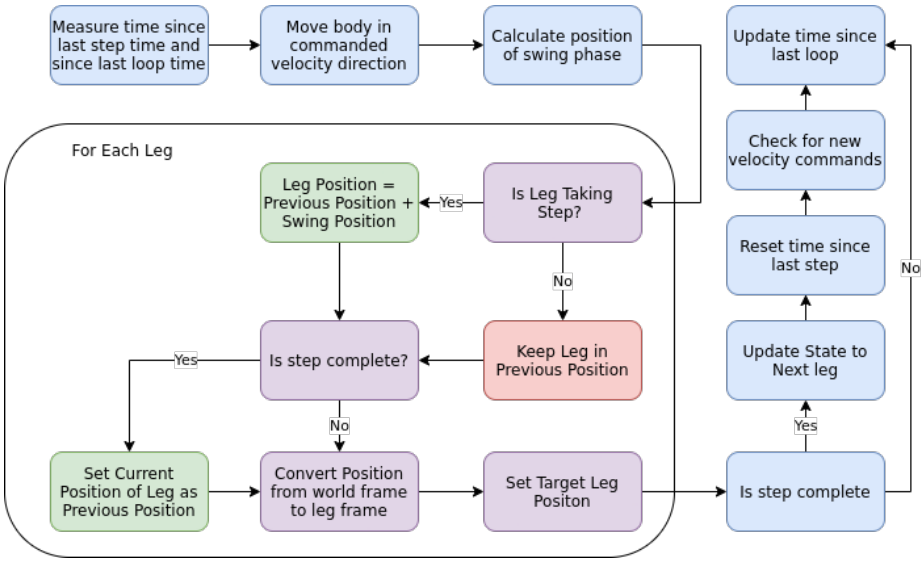


Figure 57: *Logic For Single Loop of Crawl and Trot Gait*

5.6.3 Trot

The Trot gait tries to move the robot body forward at a constant velocity while moving 2 diagonal legs forward at a time. The trot gait has a high amount of dynamic load and relies on a high step speed in order to prevent tipping

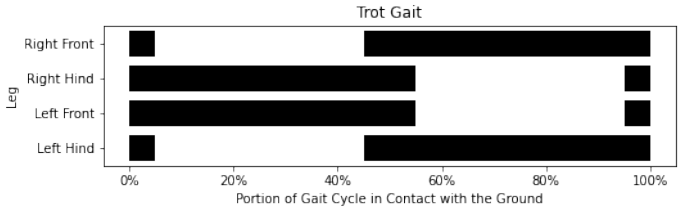


Figure 58: *Step Pattern for Trot Gait*

The Trot Gait has identical logic as the Crawl Gait (57), the key difference between the two is the Trot Gait has 2 diagonal legs set as the state at any given time. This causes the robot to try stepping with two legs at once. When updating the state the robot switches which two legs are in the step phase.

5.6.4 Intermittent Crawl

The Intermittent Crawl Gait consists of 2 major movements: First the body is moved to the center of the triangle formed by the 3 grounded legs, then the stepping leg is moved forward. The Intermittent Crawl Gait while slow is incredibly stable as it has very little dynamic load on the quadruped.

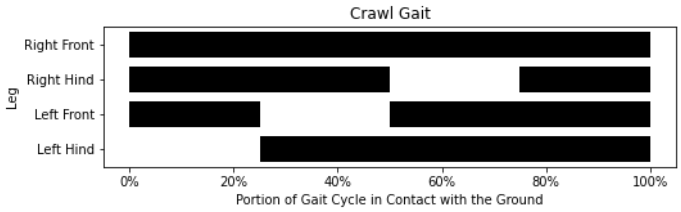


Figure 59: *Step Pattern for Intermittent Crawl Gait*

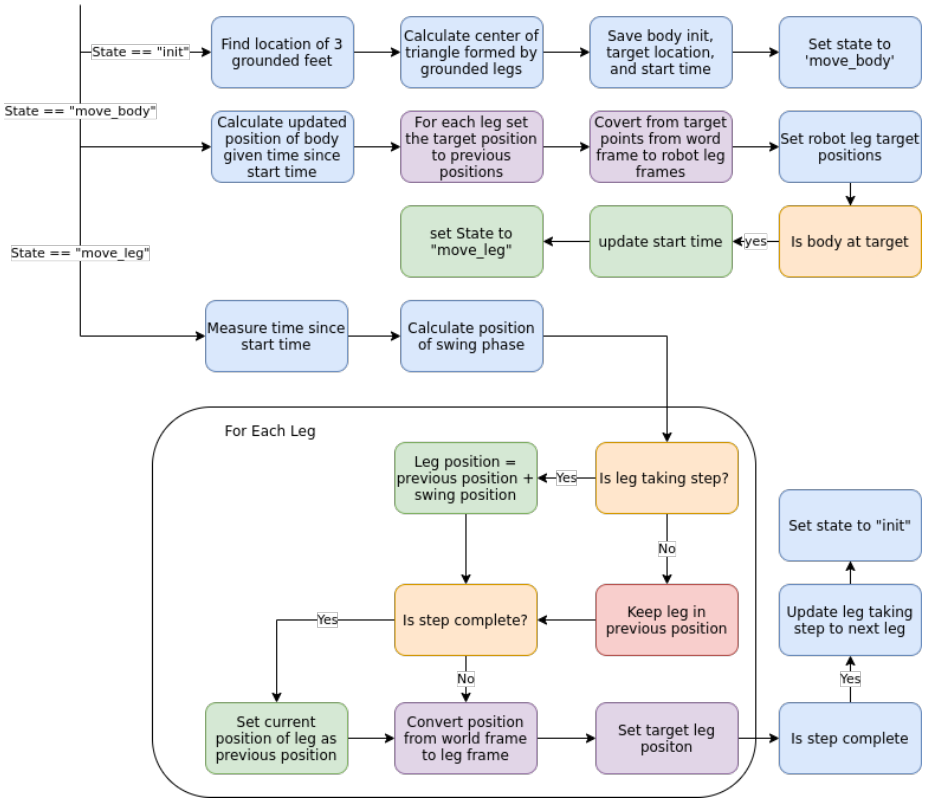
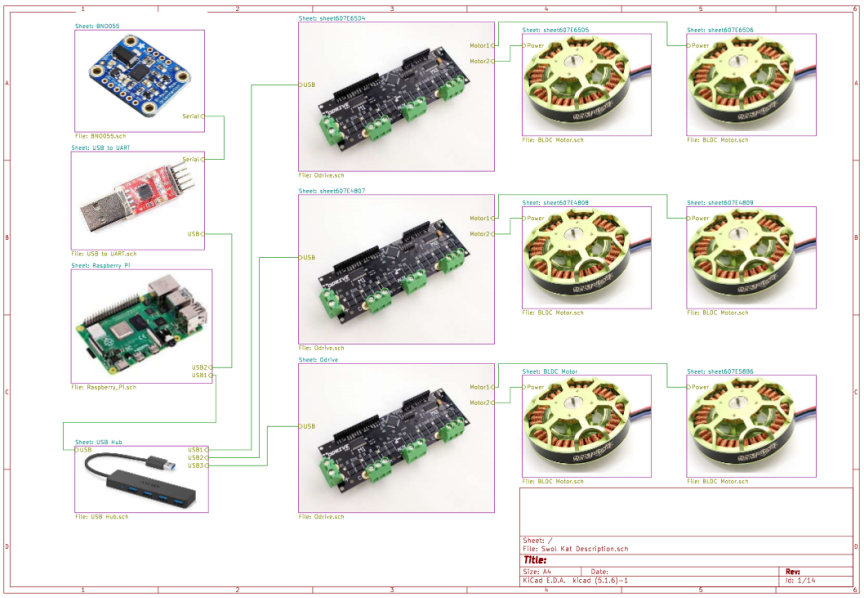


Figure 60: *Logic For Single Loop of Intermittent Crawl Gait*

5.7 Electrical



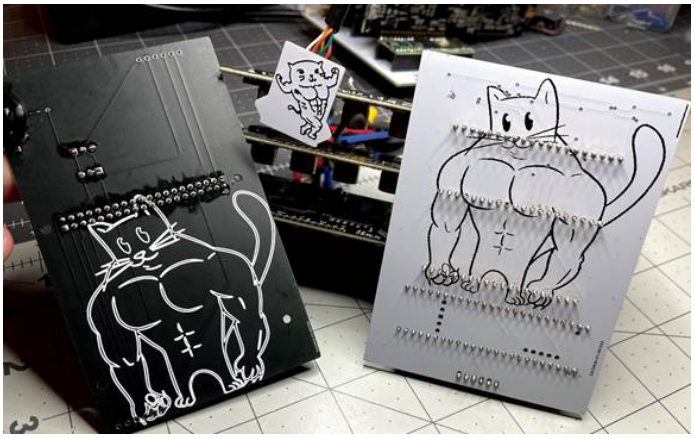


Figure 61: *Picture of Various PCB's Made*

5.7.1 ODrive Holder PCB

It was determined, based upon the planned electronics layout of the robot, that the ODrives should be mounted vertically. This would make swapping ODrives

similar to switching a graphics card. The first iteration contained features including: connectors going to the main control PCB, and 6 connectors for the encoders.

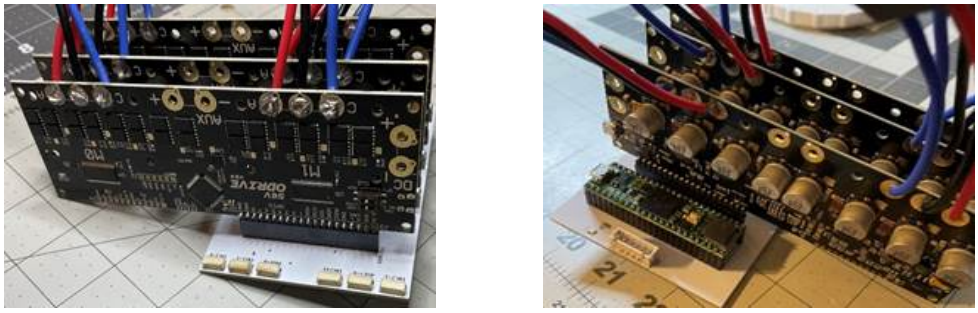


Figure 62: *ODrive Holder PCB Revision 1*

The final iteration of the ODrive PCB swaps the Teensy controller for a FT4232H mini module. This module allows a single USB to connect to 4 distinct UART channels allowing for direct communication between the Pi and the ODrives. The final iteration also included header pins to connect each of the 6 homing sensors, the ability to reset the encoders from the Raspberry Pi, as well as the capability to trigger the ODrive e-stop pins from the Raspberry Pi. The board also includes many quality-of-life features such as power LEDs and updated geometry to best fit the robot at the time the board was designed.

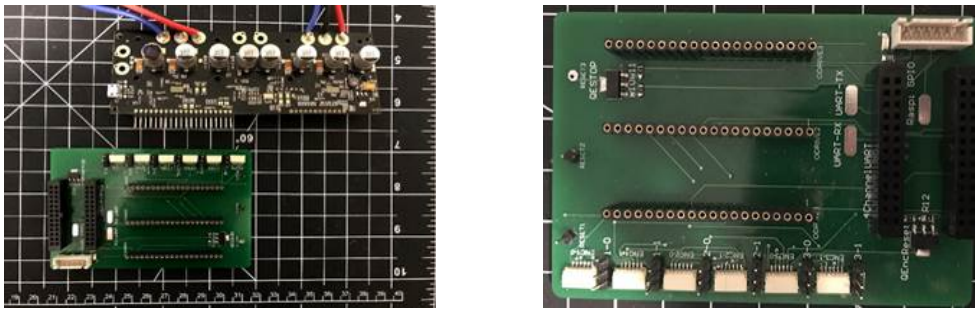


Figure 63: *ODrive Holder PCB Final Revision*

5.7.2 CPU PCB

The central controller PCB is much simpler as there are fewer components. This PCB is responsible for power distribution for the controller and daughter boards, as well as housing the Raspberry Pi. The initial design of the PCB could take either an 8S (29.6V) or 2S (7.4V) battery input to supply power to the control electronics. The PCB has two 6 pin JST connectors to connect to the ODrive PCBs. This connector has connections for 5V, 3.3V, and Ground, as well as a UART pair

and signal line used to control an LED on the controller PCB from the ODrive PCB.

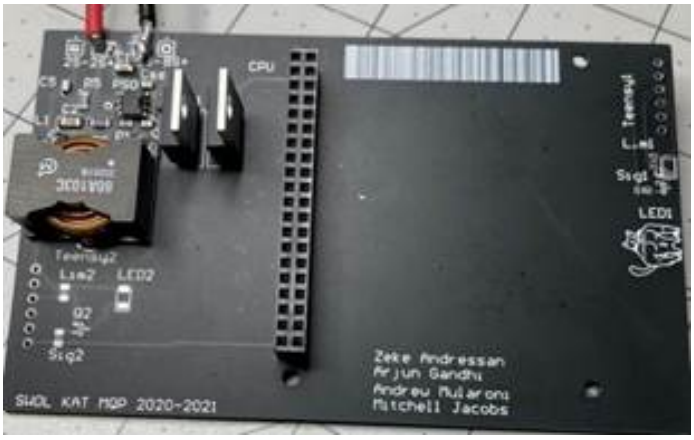


Figure 64: *CPU PCB Revision 1*

Based upon challenges with the power stage, in addition to concerns raised by the potential interference of switching regulators, the 8S input was removed on the final iteration of the board. This iteration added an input for a BNO055 dev board and used larger traces for power and ground, reducing potential heating at the maximum current draw.

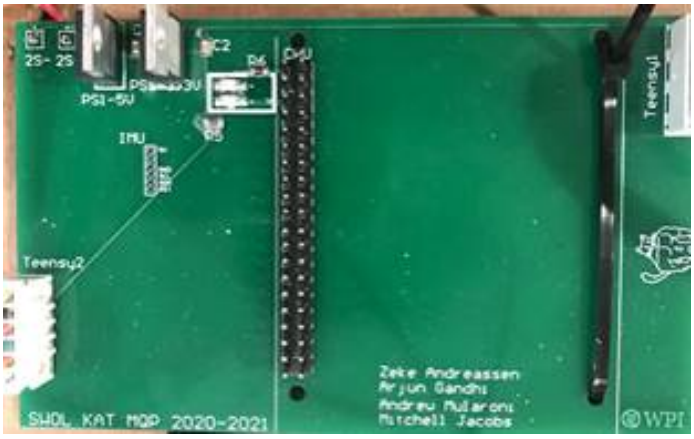


Figure 65: *CPU PCB Revision 2*

5.7.3 Encoder PCB

The encoder PCB was built with 2 main constraints in mind. The first constraint was a physical layout that is smaller than the alternative: the AS5047P dev

board. This required using the JST-SH connector, as it is smaller than the 0.1" header pins used on the dev board. The second constraint was a self centering geometry. This was created with triangular keying geometry in line with the central point of the encoder chip.

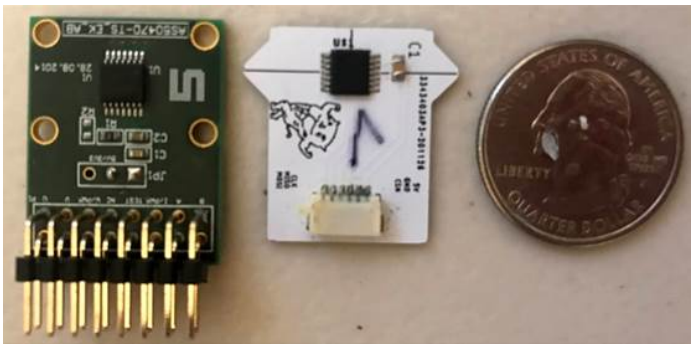


Figure 66: *AS5047P Development Board (left) Designed AS5047P PCB (center) US Quarter (right)*

5.7.4 Hall Effect Homing Sensor

The A3144 hall effect sensor is connected to the ODrive motor controllers to home the motors on startup. This sensor requires a 5V power source and outputs on a single pin that is pulled low when a strong enough magnetic field is detected. There is no pull up resistor embedded in the device, so the ODrive was configured to use an internal pull-up resistor on its GPIO pin. Futaba style three wire servo connectors are used to connect the hall effect sensors to the ODrive Holder PCB so they can be easily connected.

5.7.5 Inertial Measurement Unit

The BNO055 inertial measurement unit (IMU) was attached to the body of the quadruped so the control system can measure the rotation and acceleration of body. At first, the BNO055 IMU was connected to the Raspberry Pi through an inter-integrated circuit (I2C) connection. Due to a driver error with the Raspberry Pi I2C peripheral, reading the BNO055 would often result in errors and null values. The BNO055 was then wired to a CP2102 USB to UART converter and put into UART mode[24]. Moving to UART communication fixed this issue and allowed the program to consistently read the BNO055 measurements without any errors.

5.7.6 Battery Calculations

The batteries for the robot are chosen to allow the robot to be able to provide the peak current needed for the robot during dynamic movements. The series of batteries used for the robot have a continuous discharge rating of 75C, and

a peak discharge rating of 150C. Battery C ratings are used to determine the current that a battery of a given capacity can supply. The C rating is multiplied by the battery capacity in amp hours to determine the current the battery can supply. For the peak current, the peak C rating is used.

$$I = C * Capacity$$

An estimate of the worst case scenario for the battery would have the upper and lower extremity motors applying full torque at the maximum speed in the current limited torque region. Since the shoulder joints cannot reach this high of a speed due to the end stops of the capstan, the maximum power that they can draw is assumed to be negligible.

At the point on the torque curve where the motor is at the maximum speed in the current limited region, the motor controller is almost voltage limited so it is applying the full bus voltage to the motor. The motor is drawing the full allowable current and the full voltage, so the current draw from the battery by the motor is equal to the current flowing through the motor.

With 8 motors drawing the current limit of 64A, the current flowing through the battery would be:

$$\#n_{motors} * I_{lim} = I_{batt\_max}$$
$$8 * 64A = 512A$$

With a 150C peak battery, the battery capacity that would be needed to achieve the necessary peak current is:

$$I_{batt\_max}/C_{max} = Capacity$$
$$512A/150C = 3.413Ah$$

Since the robot should never reach peak current draw in normal operation and there was a sale on the 3Ah batteries, the team decided to go with the slightly smaller 3Ah capacity battery and be very careful to never reach this worst case scenario.

6 System Testing and Validation

6.1 Overview

Throughout development of the robot, we continuously tested components as the system grew. This allowed us to find many issues while developing the quadruped. By unit testing sections the final robot was a system with many issues already accounted for.

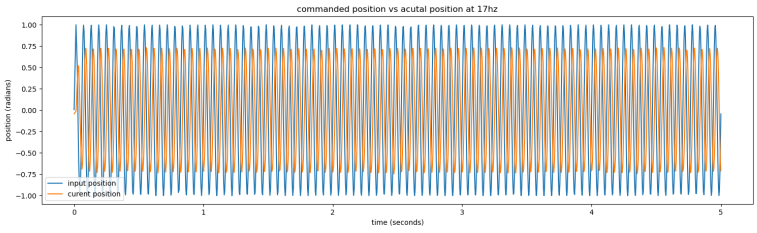
6.2 Single Motor Testing

We began by immediately building a jig to test control of a single motor using ODrive. The purpose of this testing was to prove the concept of using BLDC motors as if they were servos. Getting the motor to behave as expected took more time than originally planned, however when it was working the performance of a single motor was superb. Based upon these results we later moved on to testing control of 2 motors simultaneously to control a single leg.

6.3 Bandwith Measurement

To measure the position control bandwidth of the full system, a single unloaded motor was commanded to follow a sinusoidal trajectory with an increasing frequency until the amplitude of the trajectory that the motor actually followed was approximately 71% of what was commanded. The frequency at which the actual response is 71% of the commanded is the cutoff frequency of the system.

This cutoff frequency is a useful benchmark as it shows how well the motor control parameters are tuned and gives an indication as to how fast the main control loop must be to get maximum performance from the motors.



6.4 Single Leg Testing

Building a single leg first allowed us to test of the electronics that are required for control of the quadruped. As the shoulder joint was not yet complete, a belted system was used in place of the shoulder for testing. On this platform, the forward and inverse kinematics were validated, and the control system was tested with three motors connected at once. Some simple leg motions were tested to show that the system is capable of smooth motion.

6.5 Dual Leg Testing

Once we had two legs assembled we began testing simultaneous control of two independent legs. Testing the control system was mostly successful, however encoder errors started to appear regularly. These were able to be avoided by resetting the encoders, however the cause was unclear at the time. After further investigation, the errors seemed to be caused by some encoders being broken during

the soldering process. After switching the encoders with ones that worked, the errors mostly subsided, seemingly solving the problem.

6.6 Virtual Gait Testing

While final fabrication and assembly of the robot occurred, a test platform with only one final leg was built. This was used to test the gaits in development. By virtualizing 3 of the legs and having physical control over the single leg, the final system was able to be debugged without requiring the final mechanical assembly.

Additionally, a robot visualization software was developed to run the gaits in a virtual environment. The software draws the robot body position and all four legs in a matplotlib 3d plot. Using the visualization software, the gaits were validated on a virtual system before running them on the actual robot.

6.7 Gait Testing

Once the final robot was assembled, the first set of tests were performed while the robot was placed on a box. These tests proved that all the legs were moving properly, however encoder and ODrive issues persisted. These issues appear to be caused by noise on the encoder lines, based upon the errors prevalence on the encoders furthest from the control board. The box proved problematic as a test stand, so a system to suspend the robot with rope was set up. This proved to be a more secure method and allowed for further testing of the robot.

6.8 Suspended Testing

To test the quadruped without access to a fully equipped lab, the team had to think out of the box to create a system to suspend the robot, gradually lower it to the ground, and prevent the robot from hitting the ground if it loses balance and falls. To test the robot a master point was created to lift it. A piece of rope was tied around the front and back shoulders. A central knot was then tied. Another piece of rope was then attached to the master point that ran through a hook mounted above the robot then down to an anchor point on the floor. Using a climbing belay setup, the robot could be raised or lowered easily. This system made it much easier to test the robot without it being damaged from a fall.

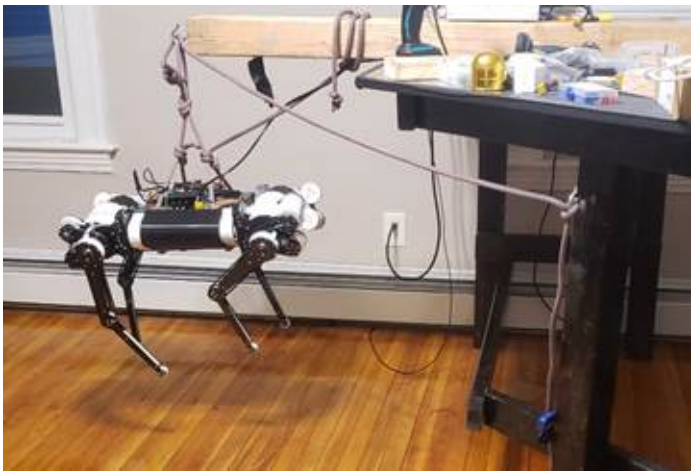


Figure 67: *Robot suspended on testing jig*

6.8.1 Lowering

Once the suspended tests proved the robot was moving as expected, the robot was slowly lowered onto its own weight while attempting to hold body position. This test proved that the robot could support its own weight in a static position. As a final test the rope was disconnected from the robot.

6.8.2 Body Positioning

Body positioning was demonstrated in the air and on the ground. It was validated that the legs moved in a coordinated way in the air, and the robot was then lowered onto the ground. Maintaining body position was proven by lightly pushing the robot and seeing that it could recover to the initial position.

6.8.3 Crawl

The crawl gait was first validated with the robot suspended. Once the leg movements were confirmed to be correct the robot was lowered onto the ground where it was able to take a few steps with the rope holding a minimal amount of the robot's weight. As more weight was taken off the rope the transmission for the lower leg began to slip leading to the rope catching the robot.

6.8.4 Trot

The trot gait was validated with the robot suspended, however based on the transmission issues present with the walk gait, it was decided to not test this gait on the ground.

6.8.5 Intermittent Crawl

Intermittent crawl was the last gate implemented in an attempt to prevent dynamic loading. This gait was successful when the robot was suspended, however when placed on the ground the static positioning could not be obtained, leading to the robot tipping over when the single leg was lifted off of the ground. Update for 2 complete steps

6.9 Encoder Problems

While setting up the robot to run with all four legs in a test environment, additional problems were discovered with the system wiring and the ODrive motor controllers. The JST-SH connectors for the encoders are not rated for many cycles of plugging in and unplugging the connector. Some of the connectors mounted on the PCB were either incorrectly connected or reached the end of product life. As a result, a few of the connectors had to be replaced as they no longer maintained full continuity across the connection, causing the ODrive to be unable to communicate with the encoder.

6.10 ODrive Problems

With the ODrive motor controllers a mysterious error would seemingly randomly occur on startup claiming that a current measurement watchdog was tripped and caused the motor controller to halt all operation. We found that simply rebooting the ODrives that had this error would fix the problem temporarily, but this is not a long term solution

7 Discussion

7.0.1 Mechanical

The final feet used on the quadruped were very slippery and did not get good traction on any non-carpeted surface. The feet were printed TPU which deforms easily but is very slippery. Due to time constraints, the team did not implement the intended solution. Other quadrupeds have cast their own urethane feet and we intended to do the same with Swol Kat. (Ben Katz Paper) The urethane we intended to use was FlexFoam-iT 23FR Flexible Polyurethane Foam.

Due to an oversight with the belt tensioning on the transmissions of the legs, the final mechanical system ran into belt slipping issues under the full weight of the quadruped. Initially the team thought these issues were coming from compliance in the 3D prints changing pulley center distances. With further examination, the HTD-3M belts that were used for the robot upper and lower extremity gearing were used out of their specified tension limits.

Tensioning devices were used to achieve slightly higher torque transmission through the belts, but they reduce the lifetime of the belts. This is neither a practical or fully functional solution.

HTD-5M belts are the next step up in the HTD belt series and are built to handle higher tension and transmit more power than HTD-3M belts. After rough calculation it is shown that these belts are not a viable solution. As such, the team recommends that in future work the quadruped be redesigned without the use of belts.

7.0.2 Electrical

The electrical and communication system proved functional for the project, however more time would have allowed for a few additional improvements. The final version of both PCBs had a few errors which could easily be fixed and help clean up the electronics further.

The ODrive PCB was rushed to manufacturing which led to quite a few oversights, from footprints being dimensioned slightly wrong, to vias not be connected to a couple of key traces. Fixing this would allow for the elimination of the USB hubs from the control system and making full use of the FT4232H mini module[25]. Additionally, more testing needs to be done with the FT4232H mini module as we broke 2 during testing and they are expensive components. These modules are also approaching end of life. We could not find any suitable alternative, however it might be simpler to implement a USB hub controller along with 3 independent USB to UART bridges. This would also reduce the form factor of the board as these components could exist in between the connection points of the ODrives.

The Controller PCB has less issues but could still benefit from another iteration. The footprint for the BNO055 IMU needs to be fixed to have proper header spacing for the dev board. It is recommended to continue using the dev board as it allows for more control over the positioning of the IMU. Additionally, the power components would benefit significantly from an external heatsink. The components have the capacity to power all the control electronics, including the Raspberry Pi (or a Jetson Nano), however there is significant temperature rise at maximum capacity.

The computational capacity of the Raspberry Pi is plenty for the current state of the project. Should additional features wish to be added, most specifically vision processing, it would be beneficial to switch to a Jetson Nano. This does not give any additional CPU capacity, however the CUDA cores would unlock additional computational capacity for specific types of tasks. If more CPU capacity is required the Teensy could be reintroduced, either in the original planned implementation, or as a separate task controller.

7.0.3 ODrives

Due to strange communication and motor errors halting progress on the rest of the quadruped, the team cannot recommend using the same ODrive motor controllers for future work.

The absolute encoder offset was unable to be saved correctly into the ODrive, so encoder calibration had to be run each time the robot restarted. Since the

encoder calibration requires the motors to freely move without obstruction, it sometimes fails while connected to the robot. A program had to be written to find the motor controllers that failed the first round of calibration and re-run the encoder calibration sequence.

The ODrive firmware does not properly implement resetting the AS5047P encoder error flag, so a custom firmware had to be developed. This cost the team development time debugging and correcting this error.

Sometimes on startup the ODrive throw error code 0x08, which only has the description "ERROR_CURRENT_MEASUREMENT_TIMEOUT"[11] This looks to be an error where a current measurement watchdog is being tripped. There are no included troubleshooting instructions for this error as it seems to be an issue with the timing of the encoder readings and ADC readings on the microcontroller.

Although connecting to an ODrive over a serial connection to a PC is a feature that ODrive claims to have, the team was not able to get a PC to connect to the ODrive over a serial connection. Scoping out the connection between the two devices shows data moving across the link, but the library is unable to produce an ODrive object that can be used in Python. This seems to be an issue with the Fibre communication library that ODrive is built on. The ODrive python library is also blocking, which causes speed issues when communicating with ODrive.

While testing tip force limiting on a single arm, a negative current limit was accidentally sent to a motor. In response, the ODrive ran the motor in full reverse and broke the arm attached to it. Knowing that this error was not caught by the firmware raises questions of other edge case scenarios where unexpected behaviors may occur.

A strange convention is used in the ODrive Python library. One problem that the team ran into is that the function to connect to an ODrive with a particular serial number only accepts the serial number as a hexadecimal converted into a string. This was not apparent from documentation of the library and took a good amount of time to resolve.

Overall, the team cannot recommend using the ODrive motor controllers for future work. Recently, a new alternative to the ODrive motor controller has emerged. The motor controller that was used in the MIT mini cheetah has become open source and some vendors are selling fully assembled versions of this controller. This controller has been proven to work in quadruped robots and the CAN bus communication protocol seems to be simpler to interface with. The team recommends that for future work this motor controller should be investigated.

7.0.4 Code

The code base on this robot is a strong foundation for the next iteration of this project. The team found the developed resources extremely helpful while programming and testing gaits. The implementation of the Virtual Joint allowed rigorous testing of all the control algorithms used on the robot. Additionally, the abstraction created by the gait super class allows for new gaits to be programmed into the robot with ease. The team was able to rapidly test and iterate through gait cycles without having to worry about how the gait was going to be run or

managed. The utility functions and classes provided quick access to many useful functions that were common among gaits. This allowed the implementation and testing of some gaits to occur within a matter of hours.

There are a variety of things the team recommends implementing moving forward. While a web-based GUI was created for Arm, and ODrive Control, due to time constraints a proper control interface for the robot was not implemented. The team recommends defining a variety of inputs to control the robot. Due to time constraints the current IMU implementation is not compatible with the multithreaded approach, for this code base to become viable for future research it is recommended that the IMU be implemented.

7.0.5 Controls

The team was able to cycle the robot through all of the developed gaits in the air. This shows great promise for the control algorithms that were developed. However, a lack of IMU feedback on the robot makes it impossible to test or implement more advanced gaits. In future versions of the code a state estimator will need to be fused with gyroscopic data to accurately track the robot's position. Additionally, a there is currently no feedback on the any of the gait algorithms. Adjusting the gaits to handle sensor feedback and adjust the robot body accordingly seems like the clear next step for the project.

8 Conclusion

In conclusion, the team must acknowledge that the initial goals were overly ambitious. Based upon the completed work this project is largely a success. The team demonstrated the robot's control system is capable of both static and dynamic walking gaits. The visualization system can be used to validate future gaits before testing them on the robot hardware. Unfortunately, the transmissions of the robot are not capable of dynamic movements, however this can be solved with future work. Replacing the ODrive controller should allow for a more robust control system and fixing the mechanical issues will result in a highly capable quadruped.

The team looks forward to seeing this project flourish as it is further developed to be a robust research platform.

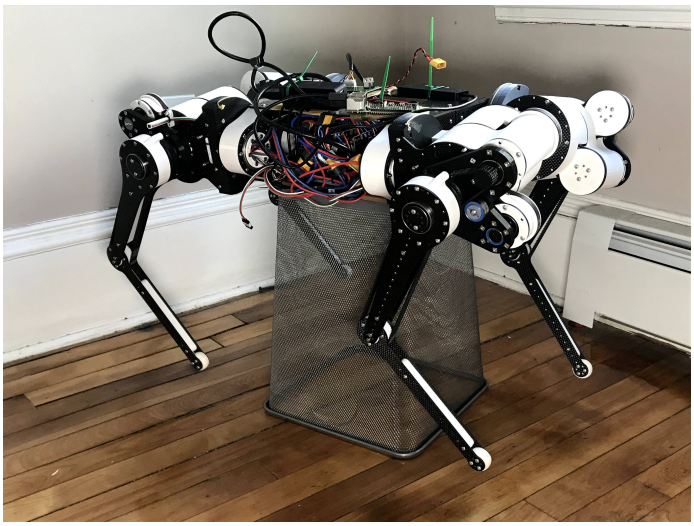


Figure 68: *Fully Assembled Robot*

9 Appendix

9.1 UI Pictures

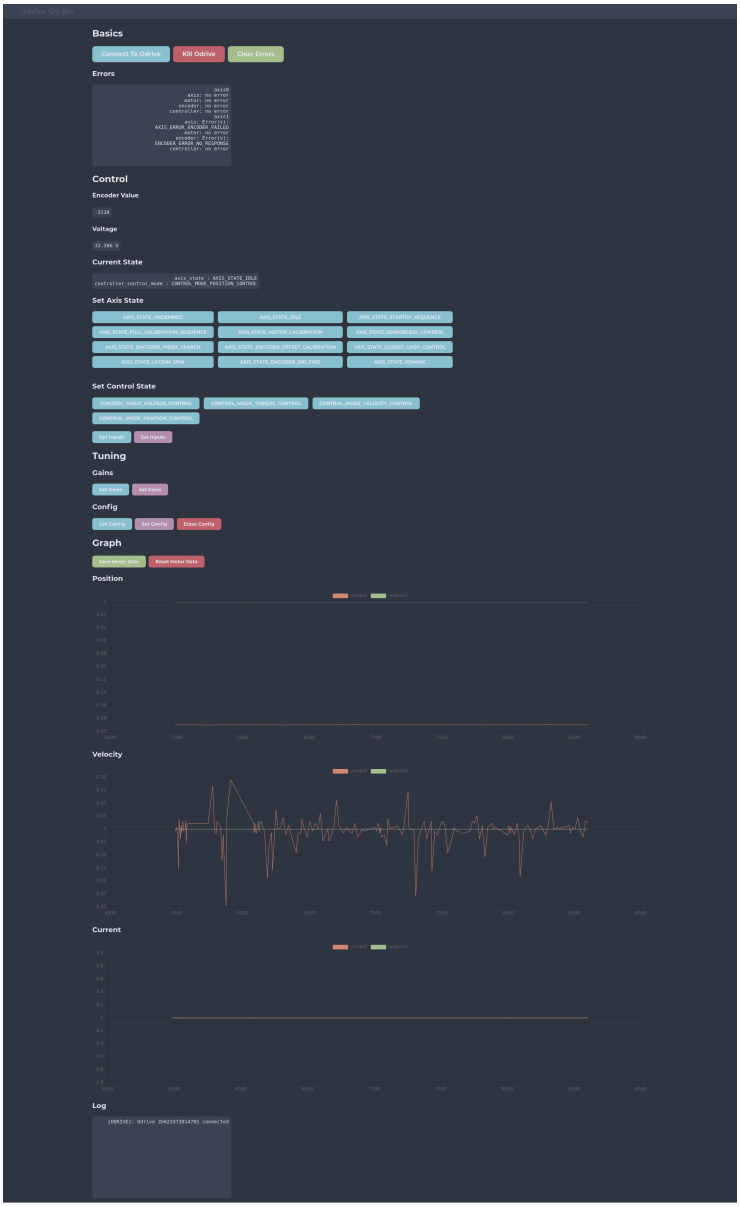


Figure 69: *Odrive Tuner UI*

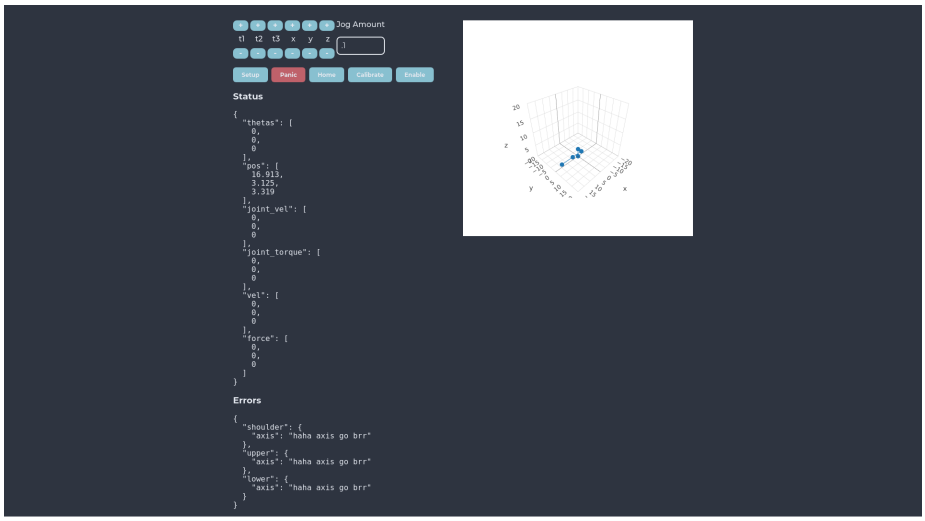


Figure 70: *Leg Control UI*

9.2 PCB Pictures

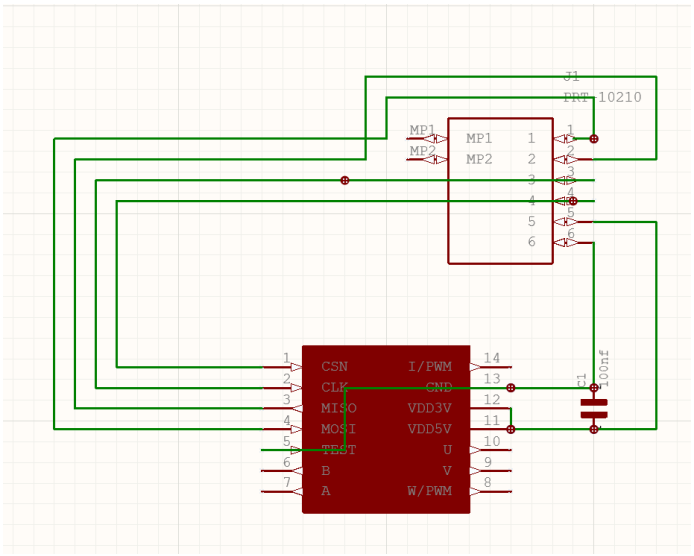


Figure 71: *Encoder PCB Schematic*

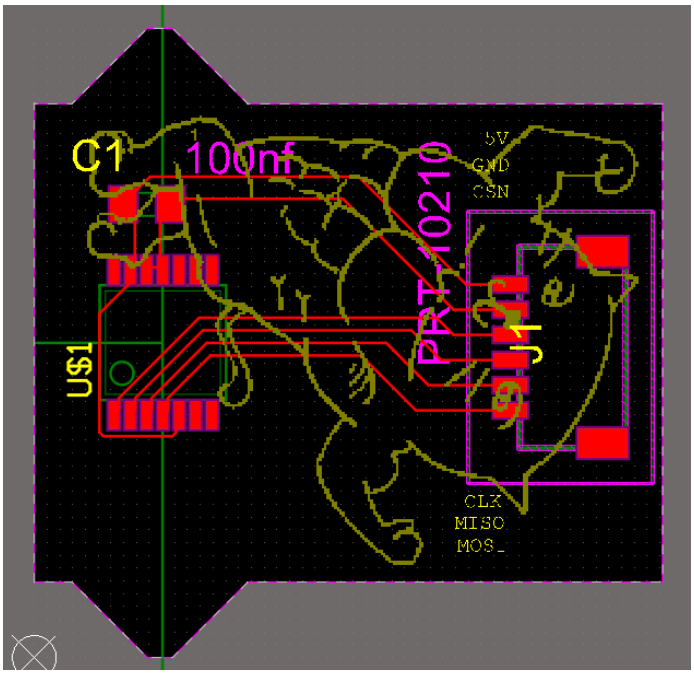


Figure 72: *Encoder PCB Layout*

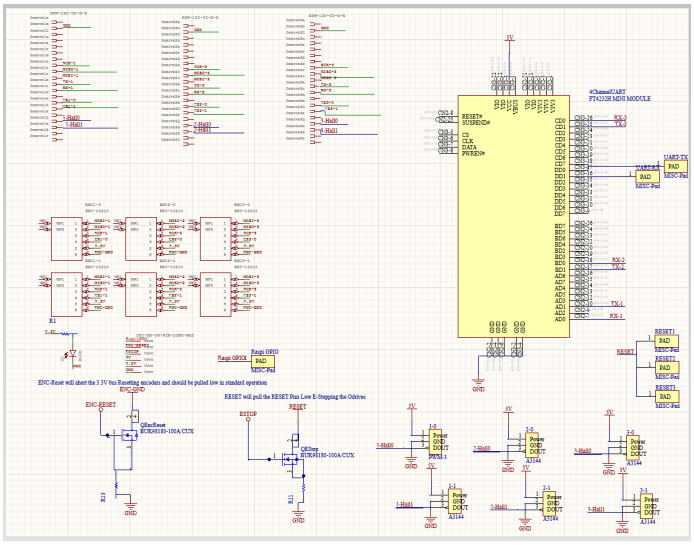


Figure 73: *ODrive Holder PCB Schematic*

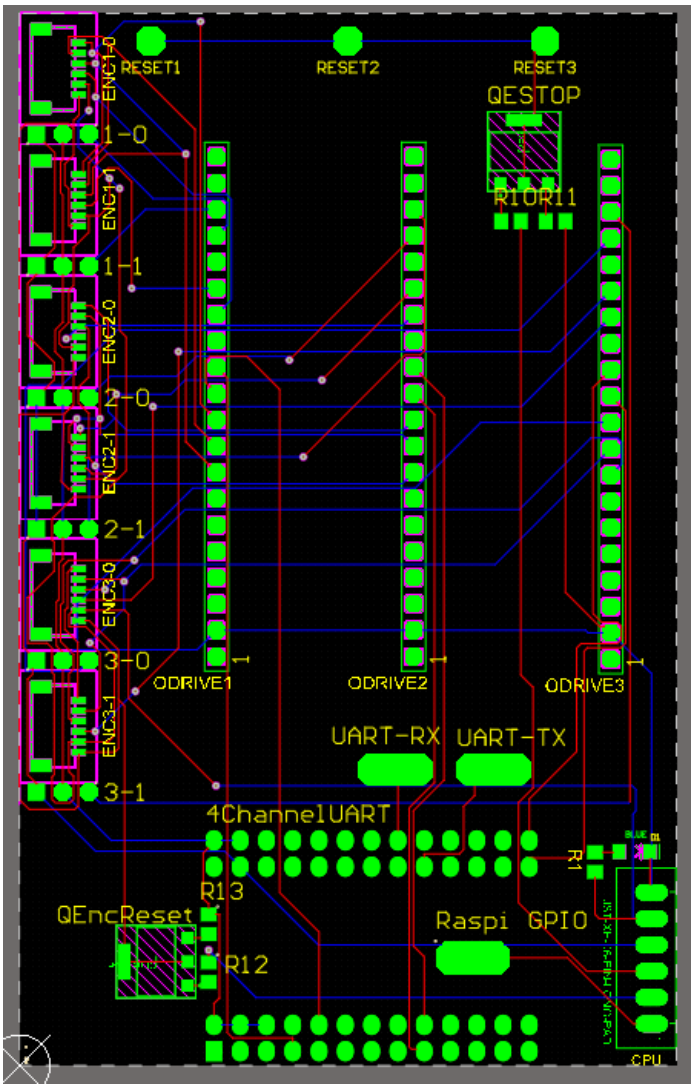


Figure 74: *ODrive Holder PCB Layout*

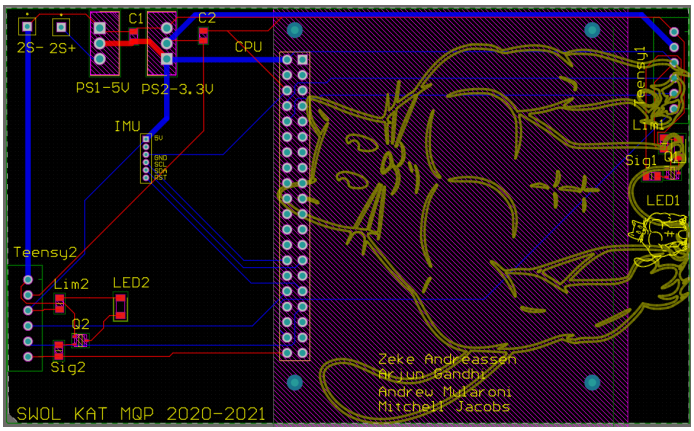


Figure 75: *CPU PCB Layout*

References

[1] W. H. Encyclopedia, "Capstan equation." [Online]. Available: http://www.self.gutenberg.org/articles/capstan_equation/

[2] "As5047p 14-bit on-axis magnetic rotary position sensor with 12-bit decimal and binary incremental pulse count for 28krpm high speed capability." [Online]. Available: https://ams.com/documents/20143/36005/AS5047P_DS000324_2-00.pdf/a7d44138-51f1-2f6e-c8b6-2577b369ace8

[3] H. Zhuang, H. Gao, Z. Deng, L. Ding, and Z. Liu, "A review of heavy-duty legged robots," Dec 2013. [Online]. Available: https://link.springer.com/article/10.1007/s11431-013-5443-7

[4] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6295–6301.

[5] "Solutions: Boston dynamics." [Online]. Available: https://www.bostondynamics.com/solutions

[6] J. E. O Rourke, V. Chernyak, T. R. Flynn, A. K. Zalutsky, and J. F. Morgan, "Sabertooth: A high mobility quadrupedal robot platform," 100 Institute Road, Worcester MA 01609-2280 USA, Tech. Rep., April 2011.

[7] A. C. Tacescu, K. Bisland, and X. J. Little, "Smallkat mqp," 100 Institute Road, Worcester MA 01609-2280 USA, Tech. Rep., April 2019.

[8] D. J. Fitzgerald, "Hydrodog: A quadruped robot actuated by soft fluidic muscles," 100 Institute Road, Worcester MA 01609-2280 USA, Tech. Rep., April 2015.

[9] A. L. Bittle and A. Martinez, “Low cost quadruped: Mutt,” 100 Institute Road, Worcester MA 01609-2280 USA, Tech. Rep., April 2017.

[10] Wikipedia contributors, “Gait — Wikipedia, the free encyclopedia,” 2004, [Online; accessed 22-July-2004]. [Online]. Available: https://en.wikipedia.org/wiki/Gait

[11] “Odrive documentation.” [Online]. Available: https://docs.odriverobotics.com/

[12] “Teensy® 4.1 development board.” [Online]. Available: https://www.pjrc.com/store/teensy41.html

[13] “Jetson nano developer kit.” [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[14] “Raspberry pi 4 model b datasheet,” June 2019. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2711/rpi_DATA_2711_1p0_preliminary.pdf

[15] “Dyneema® fiber.” [Online]. Available: https://www.dsm.com/dyneema/en_GB/our-products/dyneema-fiber.html

[16] *TECHNICAL DATA SHEET - Nylon X*, Matter Hackers. [Online]. Available: https://www.matterhackers.com/r/xCSnpt

[17] A. Mazumdar, S. J. Spencer, C. Hobart, J. Dabling, T. Blada, K. Dullea, M. Kuehl, and S. P. Buerger, “Synthetic fiber capstan drives for highly efficient, torque controlled, robotic applications,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, p. 554–561, 2017.

[18] P. I. Corke, “A simple and systematic approach to assigning denavit–hartenberg parameters,” *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 590–594, 2007.

[19] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley amp; Sons, Inc., 2020.

[20] D. Arrachequesne, May 2021. [Online]. Available: https://socket.io/

[21] “Welcome to flask.” [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/

[22] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[23] X. Zeng, S. Zhang, H. Zhang, X. Li, H. Zhou, and Y. Fu, “Leg trajectory planning for quadruped robots with high-speed trot gait,” *Applied Sciences*, vol. 9, p. 1508, 04 2019.

[24] “Bno055 intelligent 9-axis absolute orientation sensor data sheet.” [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bno055-ds000.pdf

[25] “Ft4232h mini module usb hi-speed ft4232h evaluation module datasheet,” August 2012. [Online]. Available: https://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_FT4232H_Mini_Module.pdf