April 2019

SmallKat MQP

Alex Cristian Tacescu
Worcester Polytechnic Institute

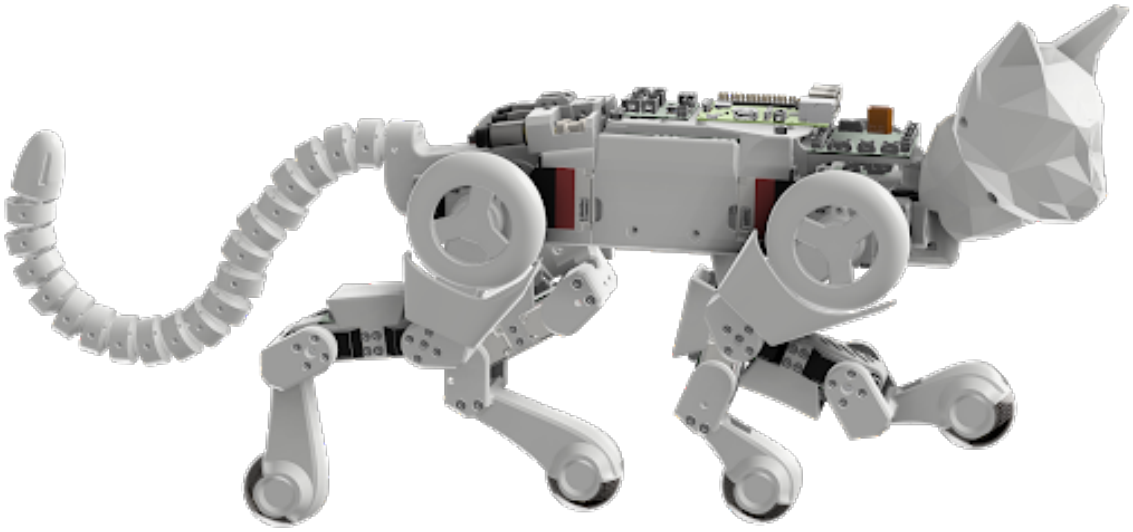
Keion Bisland
Worcester Polytechnic Institute

Xavier J. Little
Worcester Polytechnic Institute

SmallKat Major Qualifying Project

Keion Bisland, Xavier Little, Alex Tacescu

April 2019



WPI

Abstract

The SmallKat MQP is providing a quadrupedal robotic platform intending to help research and design new gaits, test sensors, and teach engineering students. Currently available quadrupedal platforms limit small companies, universities, and hobbyists due to their complexity, large size, and immense cost. SmallKat plans to change that, by providing a low-cost robotic platform with customizability and adaptability in mind. To allow for a multitude of gait designs, it is designed with 4DoF legs controlled by powerful custom servo motors, 9DoF IMUs, and a custom micro controller. The body is constructed using additive manufacturing with PLA plastic, and has a continuum tail for added body control. The higher level controller runs on a single-board computer for added performance when computing kinematics and dynamics, and controlling different gaits. Future revisions of SmallKat could look to advance the basic gaits that were developed and add new sensors like a 3D camera.

Acknowledgements

This project would not have been possible without the help and guidance of the following people:
Nicholas Bertozzi
Bradley Miller
Kevin Harrington
Loris Fichera
Gregory Fischer

We would also like to extend our gratitude to the team of engineers at Boston Dynamics who took the time to consult and advise us on this project.

Executive Summary

The field of quadruped robotics, or the study of 4-legged robots, has been inspired by the limitation of wheeled and tracked robotic systems. Walking robots generally excel in rocky uneven terrain for search and rescue and disaster relief operations. Therefore, as technology increases, quadrupeds have a huge potential to be used where other robots and/or humans can't go. However, research in walking robotics is hard to do, partially because there is not a ubiquitous platform for research and development. The few research quadrupeds include Boston Dynamics' Spot Mini and Tekken, and cost hundreds of thousands of dollars to build. SmallKat MQP's robot is designed to be a quadrupedal robotic platform to help research and design new gaits, test sensors, and teach engineering students.

In order for it to be a successful research platform, SmallKat needed to be balanced and powerful, while maintaining configurability. It also needed to be easily manufacturable and adaptable to change to allow users to modify any parts and add additional components and sensors. Therefore, we chose to design this robot to be built using additive manufacturing techniques on consumer 3D printers. For maximum adaptability, the 4-DoF legs are powered by standard sized servos (JX HV-5932MG) with custom motor controllers built into the servo casing. We chose 4 degrees of freedom since our particular configuration allowed us to customize the angle to the ground for optimal grip when walking. The molded polyurethane feet add an increased coefficient of friction with the ground and have integrated pressure sensors to measure the force vector produced when contacting the ground. Finally, we added an advanced continuum tail powered by 4 Maxon motors and a 2-DoF head for better control over balance when dynamically walking.

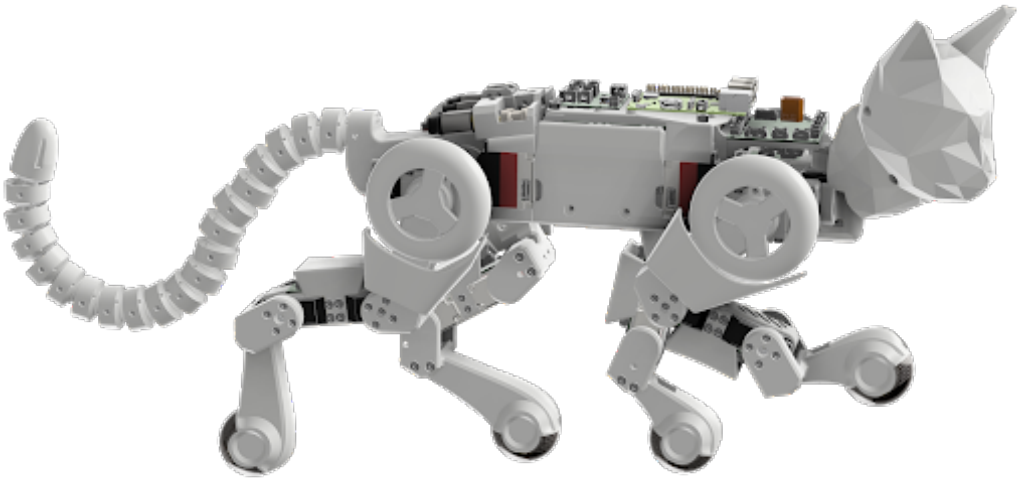


Figure 1: The SmallKat Quadrupedal Robotics Research Platform

We also developed custom electronics and a communication stack that was optimized for high-speed data transfer between our motors and sensors. The motor controllers developed were integrated inside the servo casing and ran a PID controller for the position, velocity, and torque at over 10kHz. We also developed custom foot sensors that measured the force vector from impact with the feet in order to provide feedback for advanced waking gaits in

rough terrain. In order to effectively communicate on the same communication stack, a custom charger was developed for the battery, as well as a driver and controller for the continuum tail and a 9-DoF inertial measurement board for feedback into higher level controllers. This was all tied together via a custom motherboard running a 400 MHz micro controller that connected all the boards together.

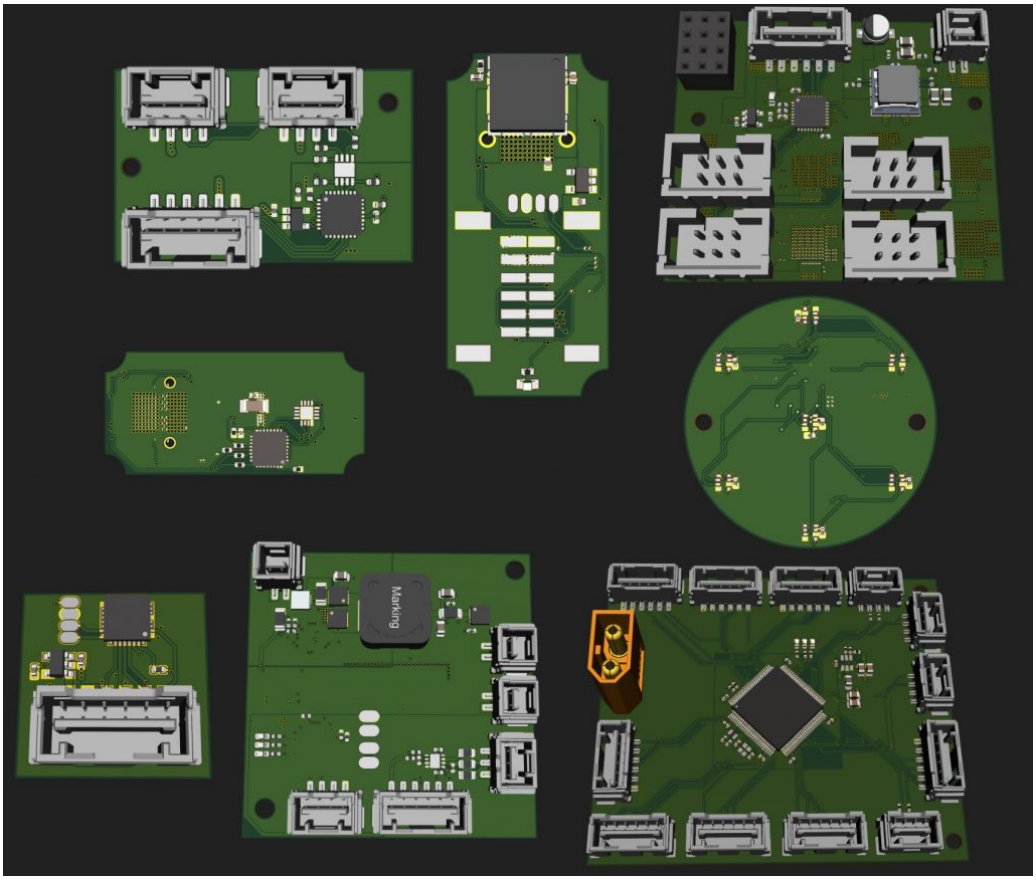


Figure 2: Electronics Integrated in SmallKat

These controllers were all tied together using a custom communication stack. SPI was used for communication between the motherboard and the IMU, the charger, the foot sensors, and the tail. To prevent having too many wires, RS485 was used to communicate with the motors in the legs and the head. Then, the motherboard relayed all this information back over USB HID to the Raspberry Pi 3 B+, which ran kinematics and dynamics calculations as well as the walking gaits. The software running on the Raspberry Pi 3 also had the capability to be controlled over WIFI to allow for wireless testing. The Raspberry Pi 3 performed kinematics and dynamics calculations using Bowler Kernel as a software platform. This allowed us to use previously developed basic walking gaits when testing. We also used Bowler Studio as a development environment, which included a simulator to test any kinematics and gaits before running on the robot. The kinematics and some dynamics were developed through this project,

and a basic walking gait was tuned and tested. We also started the development of a more complex dynamic walking gait which used a central pattern generator for trajectory generation and calculated stability using the Wide Stability Margin.

In the end, a 4-DoF quadrupedal platform was designed using custom sensors, motor controllers, and accessory boards. SmallKat also walked its first steps using the walking gait developed. In the future, we would like to see the dynamic walking gait be developed, as well as revisions for the robot as a whole, including the electronics and mechanical design. Future research can also look more carefully at the advantages of using a 4-DoF leg over a 3-DoF leg in quadrupeds.

Contents

List of Figures

List of Tables

Author	Section	Page No.
Keion Bisland	Introduction	1
	Objectives	2
	Background & Research	3-18
	Design Process : Electronics & Low-Level Software	29-34
	Development and Implementation : Electrical	37-48
	Results : Electrical	49
	Results : Overarching	50
	Conclusion & future work	51
Xavier Little	Introduction	1
	Objectives	2
	Background & Research	3-18
	Design Process : Mechanical Design	19-29
	Results : Mechanical	49
Alex Tacescu	Abstract	
	Executive Summary	
	Introduction	1
	Objectives	2
	Background & Research	3-18
	Design Process : High Level Software	34-37
	Development	Implementation: Software 60-64
	Results : Software	50

1 Introduction

The field of quadruped robotics, or the study of 4-legged robots, has been inspired by the limitation of wheeled and tracked robotic systems. These systems favor flat terrain and struggle with extremely rough, uneven or rocky terrain. On the other hand, quadrupedal systems excel in these environments. When it comes to traversing complex terrains climbing up obstacles such as stairs and rocky mountain sides there are no better systems than legged ones. Quadruped systems have huge potential because of their ability to navigate these nonuniform surfaces, this makes them ideal for exploration or providing assistance in the average home, where there may be unexpected obstacles to climb over. Using quadrupeds to explore locations wheeled robots can't is not just important here on earth but would also open a world of opportunities on other planets or moons in our strive for space exploration. In the home, quadrupeds can be used to navigate stairs or obstacles on the ground. They would allow us to create robotic assistants without having to restructure our homes or workplaces to accommodate a wheeled robot. Additionally they have many applications in the military as well. These kinds of robots are already being designed for reconnaissance or search and rescue and the United States military has contracted several of these robots to be used as robotic mules that can carry heavy items or people through the rough battlefield terrain. Since quadrupedal robots are relatively complex to manufacture, due to their abundance of components and many moving parts, the industry as a whole has had difficulty expanding. Currently, the only way to learn, experiment, develop, or create capable quadruped robots is to have the prowess of companies like DARPA (Defense Advanced Research Projects Agency) or Boston Dynamics. However, with new inexpensive processing technology and better manufacturing technologies like 3D printing, quadrupeds are on the brink of becoming more affordable than ever, allowing students and other robotics enthusiasts to explore the realm of quadrupedal systems.

Figure 3: Boston Dynamics' Spot during Military Trials with the Department of Defense

In the past few years, several MQPs have been attempted to design and build a walking quadruped for research purposes. Before beginning this project we looked into past work from the Quadruped Research Project (QRP) MQP [14], the RoboDog MQP[15], and the Hydro Dog MQP. Through analyzing their research and their project, we discovered a fatal flaw of every project was the use of 2 DOF legs and no other means of balancing. This lack of a 3rd or even 4th DOF

limited the robots ability to stabilize itself or even turn as the leg was unable to be swung in or out to regain balance. This severely limited the capabilities of the robot and would therefore be the first thing we would address when designing this new platform. This combined with the lack of adequate torque analysis gave us a deep understanding into the downfalls of the projects.

The SmallKat MQP has developed a small robotic quadruped for research and development purposes. It has 4 degree-of-freedom (DoF) legs, a continuum tail, and 3D printed body and construction. This is powered by custom servo motor controllers and 9DoF IMU sensors, all connected through an SPI & RS 485 daisy chain protocol connected to a STM32H732iit microprocessor. The higher level controller runs on a single-board computer for added performance when running kinematics and dynamics algorithms, and to control a basic walking gait. SmallKat's goal is to be used by academic institutions, corporations, or hobbyists that are interested in further developing multipedal robotics platforms. Since SmallKat is open source, anyone will be able to modify both the software and hardware to meet their needs, opening up the possibility of further research opportunities in new gaits, continuum spines, and other fields. An inexpensive quadrupedal platform like SmallKat will decrease the barrier of entry in robotics and allow for more innovation in the field.

2 Objectives

SmallKat is designed to full a void in the research and development of multipedal robotic systems by providing researchers with a platform to design new gaits, test sensors and motors, and teach the next generation of engineers. Where-as the current options only allow for a limited number of robots due to cost, size, complexity, and safety constraints, SmallKat will allow for organizations to have multiple development platforms running at the same time, allowing for comparisons of components and walking gaits.

To achieve this goal, SmallKat will be designed as an adaptable quadruped. Therefore, it must be highly modular and easily repairable. To allow for a multitude of different gaits, SmallKat will have four degrees-of-freedom (DoF) legs, differentiating it from the vast majority of other quadrupedal platforms. Each joint will be controlled by a powerful standard-sized hobby servo motor with additional sensors and components to allow for various forms of feedback: position, velocity, and torque. Finally, SmallKat will have foot sensors that triangulate the force vector being applied to each foot for contact point detection, and several inertial measurement units (IMUs) for added control over the body's linear velocity and angular acceleration. This will all be tied together with an easily adaptable open source software platform that encourages smooth development and testing of the SmallKat robotics platform.

Mechanical Objectives

General Goals

- ☐ 4 DoF legs powered by custom hobby servo sized motors
- ☐ 3D-printed rigid body with a basic head and tail for counterbalance

Reach Goals

- ☐ Continuum-style tail for added realism

- ☐ 2 DoF articulated spine
- ☐ Jaw as a manipulator

Electrical Objectives

General Goals

- ☐ Custom smart servo motors with an absolute encoder and torque sensing, tied together with over a 1khz control loop
- ☐ Custom 9 DoF Inertial Measurement Units for body control
- ☐ Embedded SPI protocol to daisy chain motors and sensors
- ☐ Powerful motherboard custom-designed to connect all the actuators and sensors to the main computer

Reach Goals

- ☐ Custom foot pressure sensors to triangulate force vector on the foot

Software Objectives

General Goals

- ☐ High speed kinematics controller running at under 5 milliseconds 95% of the time
- ☐ Basic walking and trotting gaits
- ☐ Functional remote user interface that displays debugging data

Reach Goals

- ☐ Advanced dynamic walking and trotting gaits
- ☐ Autonomous mapping using SLAM algorithms and perception sensors
- ☐ Advanced user interface with remote 3D visualization of the current state of the robot

3 Background & Research

3.1 Small(Kat) Beginnings

The SmallKat project started in October 2017 as an Independent Study Project (ISP) with Worcester Polytechnic Institute (WPI) Professor Ciaraldi. The original intention of the project was to create the smallest possible walking quadruped we could in a single term. Using a digital 7 gram servos and a custom circuit board as the back bone of the project we were able to create a 16 degree of freedom system that very much resembled a feline. The first version barely walked and while ultimately the results of this project were quite successful, it left us wanting a much better solution. During the summer of 2018 we continued the project creating a slightly larger more advanced system. This version dubbed SmallKat V2 was significantly more successful than its predecessor. It

was more reliably designed and used much more reliable servo motors and several sensors including IMUs, pressure sensitive feet and current sensing. Additionally a reliable static walking gait was created and tested allowing it become WPI's most successful quadrupedal robotic system. Two versions of SmallKat V2 were built, aptly named Crimson and Grace they have been demoed at many of WPI's undergraduate robotics demonstrations.

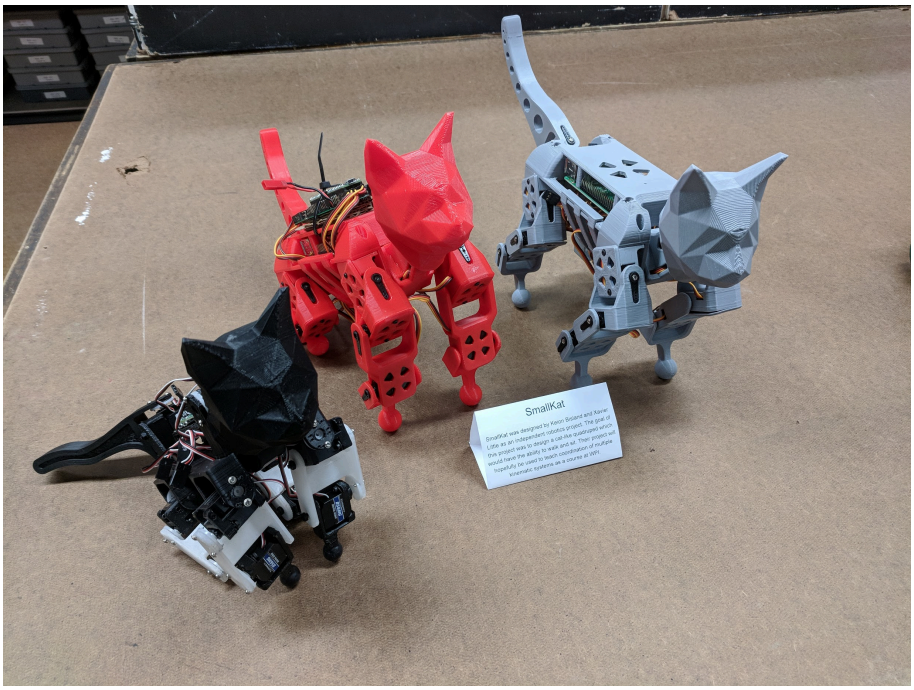


Figure 4: SmallKat Versions 1, 2 and 2.1 during an undergraduate robotics lab demonstration

3.1.1 SmallKat V1 & V2

3.1.1.1 Electrical

Developing V1 & V2 of the SmallKat project revealed a lot of nuances when it came to the electrical system. These stemmed from the lack of any form of feed back due to the limited capabilities of using PWM and hobby servos. On V1 we had a very simplistic motherboard, just enough to get the system working. The purpose of the board was to breakout the 22 servo ports available on the Teensy 3.5 micro-controller as well as regulate the 7.4-8.4v input from the battery to 5v for the Teensy and the raspberry pi zero. This board made connecting all 16 Servos much easier and cleaner. When developing the motherboard for V2 a lot more time was taken to consider the possibilities of what could be achieved from the motherboard. On version 2, USB communication between the Raspberry Pi 3 B+ and the Teensy 3.5 was used to increase communication speed. IMUs were implemented at the front and rear of the robot along with current sensors for each of the 16 motor. Integration of these sensors were our attempt to combat the lack of feedback from our system. While we still had no direct positional or velocity feed back from the motors we could still attempt to implement a rough closed loop control using IMU data or determine the torque on

the motors using the current sensing. Additional pressure sensors were added in the feet to detect when the robot was in contact with the ground. Due to time constraints these sensors have yet to be fully implemented into the control system of SmallKat V2.

3.1.1.2 Software

Embedded The Teensy 3.5 was programmed and developed through the Arduino environment. This was done in order to reduce the development time. However, this resulted in a number of issues arising. Because the target market for the Arduino environment is primarily beginners and hobbyists, there exist a great number of unnecessary checks and even more unnecessary layers of abstraction. This inadvertently limited the speed we could run the robot at. Many issues with using 16 PWM channels also arose at different points in the project as the platform is only designed to handle 12 by default, this meant a number of internal libraries had to be modified before the motors would move as expected.

Kinematics and Dynamics For the higher level control of the robot a program called Bowler Studio was used. This program was developed by WPI graduate and Undergraduate Robotics Lab Manger Kevin Harrington. All the kinematic, dynamic and trajectory planning calculations were done using Bowler and then sent to the robots using the HID protocol over USB. Due to this program being in an early Beta stage there were a number of bugs that slowed the progress of the project. However having the developer of the program as an advisor to the project many of these were resolved quickly and/or could be be worked around relatively easily. This solution proved to be quite easy to use and allowed us to quickly develop a basic static walking gait.

3.1.1.3 Mechanical Design

The mechanical aspect of the SmallKat system is centered around the servo for the system. For version 1 we used a small 7 gram digital servo. These provided a high amount of torque for their size. However they had a tendency to seize up if they were turned not under their own power. Additionally the plastic used to house them was brittle and the small flanges used to hold them in place would often break. Other design flaws of the V1 system included hard to reach bolts and cheap screws used to connect each link to the servo horn. These screws would often strip when removed which happened often because the links would have to be adjusted. Version 2 fixed many of these problems. The servos were held inside each link with a cover instead of bolting through the flanges. Additionally a thin straight lever arm was used to connect each joint to the servo. This was pressed on after the link was already installed allowing for it to be easily adjusted. From version 1 to 2 the overall design and look didn't change. The robot had 4 legs with 3 degrees of freedom and were situated underneath the body along with movable head and tail. While very similar in design to other quadrupedal systems The cat like features brought a smile to many faces.

3.1.2 SmallKat V3

The SmallKat MQP is the next evolution of the SmallKat quadruped series. Based on our prior work we are looking to create a new quadruped with more advanced systems than the previous two versions to allow us to experiment with more complicated walking gaits and control systems.

3.2 Other Quadrupedal Robots

Boston Dynamics is currently one of the leaders in quadrupedal and bipedal systems. Their robots are the source of much inspiration for people attempting to create quadruped systems. They have created a variety of quadrupeds each with different capabilities. However, the quadruped research space has been expanding quickly, and new players have arrived to the scene.

3.2.0.1 Boston Dynamics' Spot Mini

SpotMini is a quadruped system designed and built by Boston Dynamics. It is a small agile system intended to be used in a variety of applications such as the home, office or outdoors. It stands at .83 meters tall and weighs 25kg. SpotMini is all-electric and can go for about 90 minutes on a single charge, depending on what it is doing.

Figure 5: Spot Mini with Arm

SpotMini uses 4, 3 DOF legs to move itself around. It sports a 5kg, 5 DOF arm that it can use to pick things up and open doors. The sensor suite on board SpotMini and includes stereo cameras, depth cameras, an IMU, and position/force sensors in the limbs. These sensors help with navigation and mobile manipulation. SpotMini's dynamic control is unparalleled by any other quadruped system. The robot's movement is fluid and almost life like and it is able to navigate complex terrain and stairs. SpotMini can often be seen on YouTube were Boston Dynamics posts the robots new abilities and accomplishments. They plan on being able to commercial produce spot mini as a research platforms and for use in offices and homes.

3.2.0.2 MITs Cheetah 3

One of the leaders of the quadruped world is Massachusetts Institute of Technology (MIT) Cheetah 3, the latest in MIT's Cheetah line. This quadrupedal walking system weighs just about 90lbs, and

can run at a maximum of 13 feet per second. It has even demonstrated jumping onto platforms 30 inches tall and back-flipping! However, MIT Cheetah's most impressive feat is their dynamic controller. It can walk up stairs and rough terrain without the use of perception sensors, like 3D cameras and/or LIDARS. This technology was developed to help the robots maneuver successfully even when sensor data may not be perfect. MIT hopes that its Cheetah may one day provide disaster relief to areas where human access is difficult or dangerous.

Figure 6: MIT's Cheetah 3

3.2.0.3 Nybble

Nybble by Petoi is a very new player in the world of quadrupeds, announced in late 2018 on Indegogo. It is designed to be a low-cost Arduino powered robot for hobbyists. It is entirely made out of laser-cut wood pieces and 9 gram servo motors, and is powered by a custom board with an ATmega328P chip (from an Arduino Uno). Each leg has 2 degrees of freedom, forming a planar manipulator. This prevents Nybble from being able to turn effectively or to walk dynamically, absorbing rough terrain or external forces. Along with a 1 DoF tail, Nybble is only really suitable for hobbyists and low end research.[11]

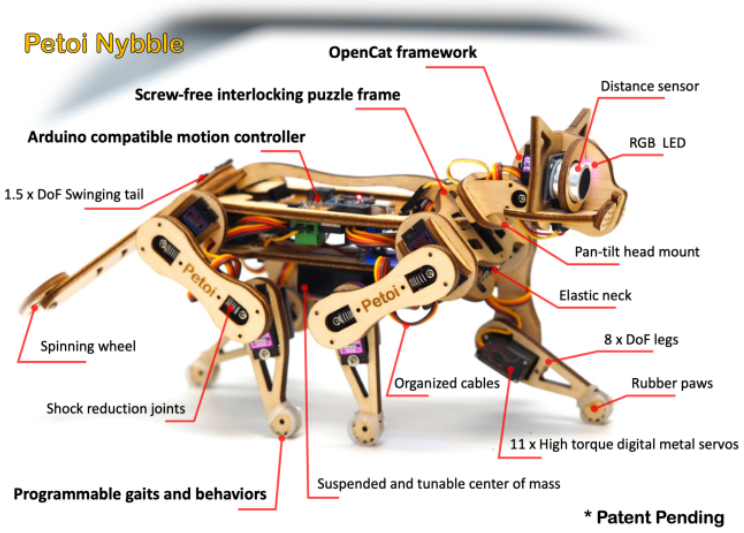


Figure 7: Nybble's components

3.3 Key Concepts

3.3.1 The Denavit-Hartenberg Convention

In mechanical engineering and robotics, DH parameters are the four parameters associated with a particular convention for attaching reference frames to the links of a spatial kinematic chain, or serial robot manipulator.

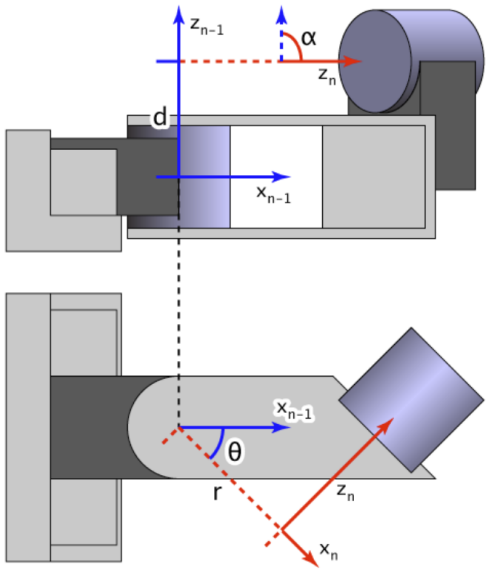


Figure 8: Visual Representation of DH parameters

Coordinate frames are attached to the joints between two links such that one transformation is associated with the joint, [Z], and the second is associated with the link [X]. The coordinate transformations along a serial robot consisting of n links form the kinematics equations of the robot. At its core D-H parameters are coordinate frames attached to the joints between two links such that one transformation is associated with the joint denoted as Z, and the other is associated with the link denoted as X. By multiplying these together you can describe a single link and its joint in a robot manipulator. The coordinate transformations along an entire serial robot consisting of n links form the kinematics equations of the robot.

$$[T] = [Z_1][X_1][Z_2][X_2]...[X_{n-1}][Z_n][X_n]$$

The coordinate transformations $[Z]$ and $[X]$, are determined by modeling the joints connecting the links as either hinged or sliding joints. By doing this, each of the joints can then be further described by a unique line S in space that forms the joint axis and define the relative movement of the two links. A typical serial robot is characterized by a sequence of these lines. For example in a six degree of freedom serial manipulator there are six S_i where $i = 1, 2, ..., 6$, one for each joint in the robot. For each sequence of lines S_i and S_{i+1}, there is a common normal line $A_{i,i+1}$. The system of six joint axes S_i and five common normal lines $A_{i,i+1}$ form the kinematic skeleton of this six degree of freedom serial robot. Denavit and Hartenberg introduced the convention that Z coordinate axes are assigned to the joint axes S_i and X coordinate axes are assigned to the common normal $A_{i,i+1}$.

This convention allows the definition of the movement of links around a common joint axis Si by the screw displacement,

$$[Z_i] = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 0 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 9: Rotation Matrix for Z axis

where θ_i is the rotation around and d_i is the slide along the Z axis. Depending on how the robot is constructed either of the parameters θ or d can be constants. The dimensions of each link in the serial chain are defined by the screw displacement around the common normal $A_{i,i+1}$ from the joint S_i to S_{i+1}, which is given by

$$[X_i] = \begin{bmatrix} 1 & 0 & 0 & r_{i,i+1} \\ 0 & cos(a_{i,i+1}) & -sin(a_{i,i+1}) & 0 \\ 0 & sin(a_{i,i+1}) & cos(a_{i,i+1}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 10: Rotation Matrix for X axis

where $\alpha_{i,i+1}$ and $r_{i,i+1}$ define the physical dimensions of the link in terms of the angle measured around and distance measured along the X axis.

As a rule of thumb, the reference frames should follow the 3 rules below.

1. The Z axis is in the direction of the joint axis
2. The X axis is parallel to the common normal. If there is no unique common normal or the axes are parallel then d becomes a free parameter.
3. The Y axis completes the reference frame in accordance to the right-handed coordinate frame system.

A screw displacement can be separated into the product of its translation and rotation about a line. This also helps better explain how this convention of forward kinematics is describing the position of each joint. By doing this the Z axis transformations can be described as;

$$[Z_i] = Trans_{Z_i}(d_i)Rot_{Z_i}(\theta_i) \tag{1}$$

And the X axis transformations are;

$$[X_i] = Trans_{X_i}(r_{i,i+1})Rot_{X_i}(\alpha_{i,i+1}) \tag{2}$$

Using this notation, each link can be described by a coordinate transformation from the concurrent coordinate system to the previous coordinate system simply by multiplying each of these transformations together resulting in;

$$^{n-1}T_n = Trans_{z_{n-1}}(d_n) * Rot_{z_{n-1}}(\theta_n) * Trans_{x_n}(r_n) * Rot_{x_n}(\alpha_n) \tag{3}$$

Breaking down each of these transformations into there matrix transformations we get:

$$Trans_{Z_{n-1}}(d_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{Z_{n-1}}(\theta_n) = \begin{bmatrix} cos(\theta_n) & -sin(\theta_n) & 0 & 0 \\ sin(\theta_n) & cos(\theta_n) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Trans_{X_n}(r_n) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Rot_{X_n}(\alpha_n) = \begin{bmatrix} cos(\theta_n) & -sin(\theta_n) & 0 & 0 \\ sin(\theta_n) & cos(\theta_n) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{n-1}T_n = \begin{bmatrix} cos(\theta_n) & -sin(\theta_n)cos(\alpha_n) & sin(\theta_n)sin(\alpha_n) & r_n cos(\theta_n) \\ sin(\theta_n) & cos(\theta_n)cos(\alpha_n) & -cos(\theta_n)sin(\alpha_n) & r_n sin(\theta_n) \\ 0 & sin(\alpha_n) & cos(\alpha_n) & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 11: Breakdown of transformation matrix

At this point we can introduce the actual DH parameters They were briefly mentioned above when describing the screw displacements but are described below;

- d: offset along previous z to the common normal
- θ: angle about previous z, from old x to new x
- r or a: length of the common normal a.. Assuming a revolute joint, this is the radius about previous z.
- α angle about common normal, from old z axis to new z axis

Combining everything we have discussed, the screw displacement, transformation matrix and DH parameters we get a matrix that looks like the following: By placing reference frames using the rules above and using this matrix, the forward kinematics of any serial manipulator can be easily determined where the x,y,z of the end effector is described as the 3x1 matrix $[T_x; T_y; T_z]$ in the result of the final matrix;

$$T_{(i-1)i} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i)cos(\alpha_i) & sin(\theta_i)sin(\alpha_i) & r_n cos(\theta_i) \\ sin(\theta_i) & cos(\theta_i)cos(\alpha_i) & -cos(\theta_i)sin(\alpha_) & r_n sin(\theta_i) \\ 0 & sin(\alpha_i) & cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 12: Transformation Matrix via DH Parameters

$$M_{n-1,n} = \begin{bmatrix} R_{xx} & R_{xy} & R_{xz} & T_x \\ R_{yx} & R_{yy} & R_{yz} & T_y \\ R_{zx} & R_{zy} & R_{zz} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

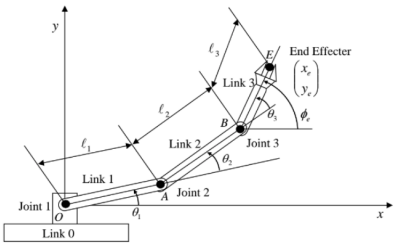
Figure 13: Resulting matrix

3.3.2 Kinematics

Kinematics is often referred to as the geometry of motion and describes the motion of points, bodies and systems of bodies without considering the forces that describe the motion. A kinematics problem begins by describing the geometry of the system and declaring the initial conditions of any known values of position, velocity and/or acceleration of points within the system. Then, using arguments from geometry, the position, velocity and acceleration of any unknown parts of the system can be determined. Using kinematics there are a variety of different calculations that can be done to help create an accurate and useful model of a robot.

Forward Kinematics

Forward Kinematics refers to the use of kinematic equations to determine the position, usually in x, y, z coordinate space, of an end effector given specific joint parameters with relation to a specific reference point. In the case of a serial manipulator, this is very useful in understanding where the end of the robot will be if given a set of joint values. There are many different ways of calculating forward kinematics, however one of the most universally accepted and used methods is through the use of Denavit–Hartenberg parameters (DH parameters).



$$x_e = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$
$$y_e = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3)$$

Figure 14: Forward Kinematic equations for a 3 DOF system

Inverse Kinematics

Inverse Kinematics is, as the name makes it out to be, the inverse of forward kinematics. Therefore, the equations obtained tell us position in joint space when given a position in task space. In some cases, a redundant link is used, such as in a 3-DoF planar manipulator. To solve for this case, another measurement in task space should be used. Generally, there are 4 main ways to solve for Inverse Kinematic Equations: the Algebraic method, the Geometric method, the Jacobian method & the Heuristic method.

Algebraic Method The algebraic method is conceptually the simplest of the 4 methods, but also causes the most amount of problems. To use this method, just find the algebraic inverse of the Forward Kinematic equations. This is relatively simple to compute in small applications with few degrees of freedom. However, when the manipulator becomes more complicated, it often fails to produce usable equations. Therefore, this method is useful for hand-computing 2-DoF manipulators.

Geometric Method The geometric method is another useful way to calculate slightly more complex inverse kinematic systems. Using geometric concepts, the Inverse Kinematic equations can be calculated, starting from the position in task space to find the angles of each joint (ie the position in joint space). Some of these geometric concepts will include the Law of Cosines, the Law of Sines, and the Pythagorean theorem. This method is usually used for more complex 3 dimensional situations, usually with 4-DoF or fewer.

Jacobian Method The Jacobian method of approximating Inverse Kinematic equations is a rather simple, yet effective concept. It's backbone is based off of the linear algebra concept of changing rank: $\mathbb{R}^n \to \mathbb{R}^3$ where n is the number of degrees of freedom of the manipulator. Given that $p_0 = p(x_0)$ as the initial position and $p_1 = p(x_0 + \Delta x)$ is the final position, the Jacobian method iteratively computes an estimated Δx and minimizes the error given by $p(x_0 + \delta x_{estimate}) - p_1$. For small Δx vectors, the following equation derived via series expansion from the position function gives: $p(x_1) \approx p(x_0) + J_p(x_0)\Delta x$, where J_p is the Jacobian matrix (see more in Section . Taking the Moore-Penrose pseudoinverse [10] of the Jacobian and rearranging the equation results in: $\Delta x \approx J_p^+(x_0)\Delta p$, where $\Delta p = p(x_0 + \Delta x) - p(x_0)$. This will give a very rough estimate of the

$$\Delta x_{k+1} = J_p^+(x_k)\Delta p_k$$

Figure 15: Jacobian Method Equation

desired Δx vector. To get a more accurate value, a line search can be used. The final recursive function is given in Figure 15. [8]

Heuristic Method Another approximation method for calculating Inverse Kinematics is the Heuristic Method. The most popular heuristic algorithms include the Cyclic Coordinate Descent (CCD) [12] and the Forward And Backward Reaching Inverse Kinematics (FABRIK). [1]

3.3.3 Basics of Gaits

What is a gait

Gait is the pattern of movement of the limbs of animals. Most animals use a variety of gaits, selecting gait based on speed, terrain, the need to maneuver, and energetic efficiency. [6]

Types of gaits

There are two general categories of gaits under which all others fall, these are Symmetric and Asymmetric gaits. A symmetric gait is defined when the left and right limbs alternate. where as an asymmetric gait, the legs move in unison. Examples of these gaits can be seen below.

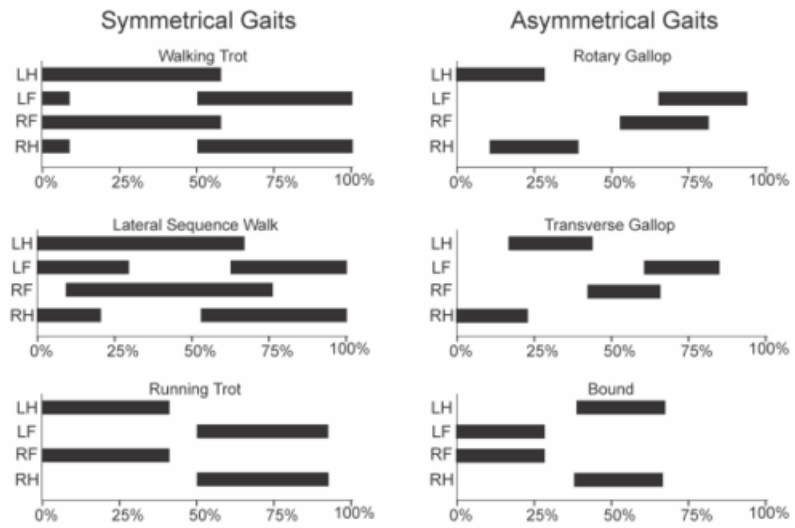


Figure 16: Graphic showing motion in types of gaits

3.3.4 Dynamic Walking

When it comes to designing dynamic walking gaits, researchers have looked toward the natural world for inspiration. Dynamic walking has been the focus of researchers everywhere, from the professors and students at MIT who worked on Cheetah 3 [2] to the engineers of Spot Mini. [19] A popular method to dynamic walking is based on a Central Pattern Generator (CPG) and a mix of predefined responses and reflexes. This is similar to nature, where animals (both bipedal and quadrupedal) use a pattern generator which is overrun by reflexes when balance is lost.

Central Pattern Generator (CPG) Central Pattern Generators (CPG) is a control method that generates trajectories based on loosed-coupled oscillators, similarly found to many neural circuits in animals. Essentially, CPGs are a biological neural network that produces rhythmic output patterns without sensor input given a signal to start.[7] It is what makes breathing, swallowing, chewing, and even walking work in humans and other animals. The same concept can be used in robotics to provide a trajectory for basic trotting gaits, and can be very stable against uncertainties in environments, like an obstacle in the way. There are two different models that can be used: biological neural networks and coupled oscillator networks.

Biological neural networks use a control system similar to our neural networks in our bodies. An example of a model is the Rulkov map type neuron. This model produces a constant oscillating wave given some input parameters by using two neurons that can oscillate when connected together. Although this method is very efficient, it exhibits spiking dynamics similar to that in galangia, [16] as shown in Figure 17

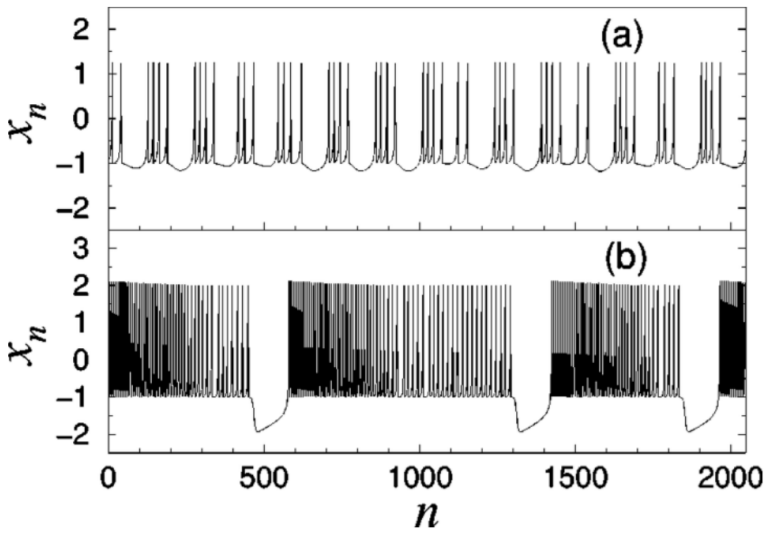


Figure 17: Wave forms generated using Rulkov Map type neurons, which display the spiking-bursting behavior [16]

Another method is to use a mathematical model, such as the Kuramoto oscillators or harmonic oscillators. Such models simplify the system as a whole and make outcomes more predictable, but

different from natural CPGs. However, it is still very possible to replicate much of what biological neural networks can make.

Although CPGs by definition are open loop, i.e. they do not need sensory feedback, it can be incorporated to modify the parameters of the CPG.[7] For example, a snake robot might need to change its oscillating behavior depending on the medium it is traveling on. This can be inputted into a properly designed CPG to allow for easier transition between gaits.

Wide Stability Margin (WSM) One of the best ways to measure the stability of a robot is to use the Wide Stability Margin method. The WSM method uses a key concept in physics: balance is based on whether the center of gravity is outside the points of contact. The procedure is very simple: project the points of contact on the ground (feet in the case of a quadruped), along with the center of gravity, and see if the center of gravity is outside the polygon drawn by the projected points of contact, as seen by the example in Figure 18.[5] The WSM factor can be determined by measuring the shortest distance between the projected center of gravity and the polygon made by the projected points of contact.

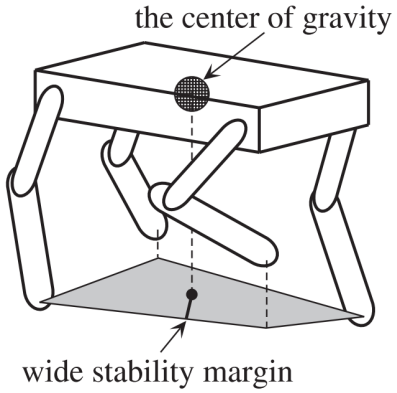


Figure 18: Example of Wide Stability Margin on a quadruped robot [5]

3.3.4.1 Basic Requirements for Dynamic Walking

In order for a robot to be able to dynamically walk, it has to follow a few criteria: [5]

- ☐ The robot should have the ability to move the leg forward/backward and sideways.
- ☐ The gait used should have a short period in order to maximize the WSM.
- ☐ The swinging leg should be able to move forward during the swinging phase of the gait.
- ☐ The leg should have a high coefficient of friction with the ground (no slipping)
- ☐ Angular movement should be minimized while the legs are moving during a gait
- ☐ The movement between the body and legs should be kept separately regardless of irregular terrain

3.3.4.2 Responses and Reflexes

Using a natural model requires that the robot also has reflexes and responses. Reflexes are predefined movements that are used in emergency, for example when a robot is falling over. Responses, however, are adjustments in the gait to reflect changes in the environment, like when a robot is walking on tilted terrain. Both reflexes and responses are necessary adjustments when developing a walking gait using nature as a model. Usually, software will contain pre-populated conditions and responses/reflexes for those conditions in order to minimize the response time of an event.

	sensed value or event	activated on	necessary conditions
flexor reflex	collision with obstacle	sw	[2]
stepping reflex	forward speed	sw	[4]
vestibulospinal reflex/response	body pitch angle	sp	[4]
tonic labyrinthine response	body roll angle	sp&sw	[3], [4], [5]
sideways stepping reflex	body roll angle	sw	[4]
corrective stepping reflex/response	loss of ground contact	sw	[3], [4]
crossed flexor reflex	ground contact of a contralateral leg	sw	[2]

Figure 19: Example of predefined reflex and response table from Tekken 2 [5]

3.4 Interviewing Boston Dynamics

During the course of the project we reached out to Boston Dynamics and arranged a conference call between our team and a group of their engineers from the spot team. We chose to reach out tho them as they are the current leader in the quadrupedal robotics field and would therefore have the most insight into the intricacies and problems that arise in the development of such a platform. We entered the meeting with a series of questions and subordinate related questions. This list comprised of:

- Why was the decision mate to pursue 3DOF legs instead of 4DOF as most quadrupedal animals have.
  - Why did you choose to abstain from a head and tail for balance and dynamics
  - What are some of the different gaits you have developed and why did you choose them
  - How have you obtained such a smooth waling gait.
- How are your trajectories and kinematics calculated pragmatically
  - Is a real time physics engine used
  - Are Jacobians calculated at run time
- What kind of sensors are used on Spot mini
  - How is each sensor integrated and used for corrections
  - recommendations on where to locate each kind of sensor
- How importance is torque control in dynamic gaits
- What was the minimum control loop speed you found was necessary for smooth operation

These questions stemmed a very informative conversation between ourselves and the team of engineers which comprised of 2 Electrical engineers and 1 Mechanical engineer, sadly no controls engineers were able to participate however those in attendance were very knowledgeable and were able to answer the majority of our controls related questions. From this conversation we gained a great deal of insight into the tough process that went into developing Spot mini as well as many previous robots in the Spot and big dog series.

We started off by asking about the decisions that led to them choosing to only use 3 DOF legs instead of using 4 DOF legs as most quadrupedal animals in nature do. To this we were promptly responded when possible keep the mechanism and calculations as simple as possible, they had tested 4DOF legs on previous robots and found they generally avoided using them and that everything they needed to do could be achieved using 3 DOF, In addition using only 3 DOF also cut down on the wiring, and weight in the legs and therefore lowered the inertial mass of the robot making it over all easier to control. They continued on to speak to the benefits of adding a joint near the center of the body in order to act as a spine which would be much more useful when it cam to highly dynamic gaits and motions allowing for the body to fold more and allow for a smoother transference of momentum.

This followed into the choice of force sensing, For the project we chose to use an array of barometric pressure sensors encapsulated in polyurethane to calculate a pressure vector. The engineers at Boston dynamics had actually tested a similar kind of sensor and had a great deal of issues using them of their robots as the repetitive motion in walking would either damage of change the calibration of the sensor. In combination with this the temperature of the environment had a great effect of the polyurethanes ability to transmit the applied force to the sensors. This combined with the damage that would get caused to the polyurethane during outdoor testing led them to decide not to continue using this method of sensing. Instead they choose to sense force at the actuator. Their recommendation was to use current to propagate torque at the motor as they used a combination of this and a custom force sensor that they could not speak too much to due to it being part of their ”robot secret sauce”.

Following this we asked about the development of gaits and their choice of a trot gait in which the legs are moves in a FR, BL, BR, FL sequence instead of a more natural pace gait in which the legs are moved in a FR, BR, FL, BL sequence. Their response was different to what we had expected, in end Spot is able to perform a pace gait however it is less stable than the trot gait. The trot gait was the first successful gait the team was able to develop when working on the original spot project and therefore has had the most development put into it. Further gait development came with difficulties, the development of a bounding or galloping gait was severely hindered by the lack of an effective spine, without this the body would form an oscillating system making the robot unstable. However when mentioned the concept of a continuum spine which would result in the greatest mobility they greatly recommended against it, this was due to the uncertainty and inability to guarantee the position of the system for a given input. For all dynamic motions such as recovering from a push or jerk, they rely heavily on the IMU which in a ridged body robot they recommend placing it directly over the center of the robot however for a split body or one using a spine they recommended placing one on each segment in order to capture the motion of each independent system.

Leading from this to kinematics and trajectory planning allowed us to learn that they do use both a simulation and a real time physics engine as well as calculation all necessary jacobians at run time. This led us to asking details of their control feedback and they recommended getting a high level controller running at 1 kHz for a smooth trajectory and a low level controller as fast as

possible in order to have a stable PID loop and remove all jittering and jerkiness from the system.

4 Design Process

4.1 Mechanical Design

4.1.1 Design Requirements

- ☐ Lab appropriate size
- ☐ Additive Manufacturing method with polyactic acid (PLA)
- ☐ Modular & Adaptable to change
- ☐ High coefficient of friction with the ground
- ☐ Able to move leg both forward/backward and sideways
- ☐ Powerful and speedy motors that can sustain the weight of the robot
- ☐ Head and tail that can be used to shift center of mass of the robot
- ☐ Aesthetically pleasing

4.1.2 Mechanical Design Process

The mechanical design of SmallKat was very important. The robot needed to be balanced and designed properly so that the motors would be able to move the whole system and be able to properly walk. It also needed to be designed in a way that was easy to modify or change out components if users wanted to change something or needed to fix a part that had broken. Before actually making the robot we needed to go do some research, testing and analysis. Below are the three steps that we took on top of the research we had previously done.

- ☐ 3 DOF vs 4 DOF
- ☐ Motor Selection and Testing
- ☐ Kinematics and Analysis

4.1.3 3-DoF vs 4-DoF

The leg design was the most important of decisions when designing SmallKat mechanically. The first decision was to decide how many degrees of freedom our system would have. More degrees would allow for more flexibility and give users more options for gait designs. However, too many degrees of freedom per leg would not only complicate control algorithms, but also increase weight per leg, which would in turn increase rotational and linear inertia, add more strain to the motors, and decrease dynamic gait performance. It is important to note that SmallKat needed at least 3 degrees of freedom to allow for forward and sideways movement, necessary for dynamic walking. Therefore, the decision was between 3 and 4 DoFs.

Early on in our design process, we spoke to Boston Dynamics (as talked about in Section 3.4), and one of the questions we asked was their opinion on 3-DoF vs 4-DoF legs. In their research, they saw the design simplicity to be the major benefit of 3-DoF legs, so almost all of their quadrupeds were 3 degrees of freedom; a simpler design means less can go wrong for their customers. However, the engineers did admit they didn't look into different 4-DoF leg designs.

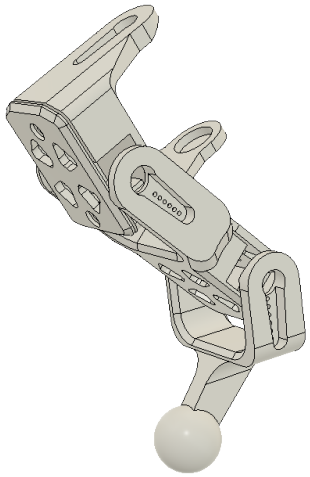


Figure 20: Planar 3-DoF Leg Design

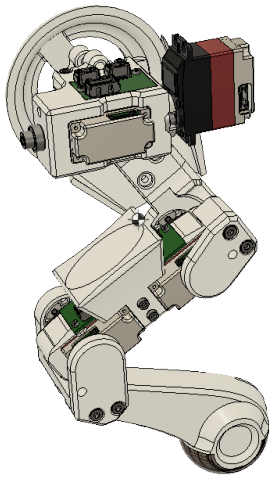


Figure 21: Planar 4-DoF Leg Design

After some deliberation and research, we ultimately decided to take a different approach to a 4-DoF leg: a 3-DoF planar manipulator with a rotation axis in the hip, as depicted in Figure 21. This would give the platform the capability to change the angle of contact with the ground, potentially improving performance on irregular terrain. This novel leg design, we call a planar 4-DoF Leg, is different from any other quadruped we found in our research. Our initial concern of not having enough speed and power in the motors we selected were taken care of after some testing (see Section 4.1.4). However, the main reason we chose 4-DoF legs instead of the 3-DoF variant was the adaptability it gave the users of our platform. Additionally because it has not been done before in any of the quadrupedal robot systems we have found it offered a unique opportunity to research the difference between 3 DOF and 4 DOF systems.

4.1.4 Motor Selection and Testing

One of the most important parts of any robot is it's motors. We considered many different motor solutions for this project, including standard hobby servos, planetary geared motors, and "smart" motors from companies like Dynamixel and Maxon. Our decision - standard hobby servos - was mostly based on the selection and standard that has been formed in the hobby industry. If a user wanted to replace their motors with weaker or more powerful version, he or she could, assuming the servo they purchase are the same size as the ones we designed around. This gives the users the option of motors to use based on their needs and budget.

4.1.4.1 Motor Options

Based on the size, we found a few options of different motors we can use. These include a standard hobby servo (JX-Servo HV-5932MG) and 2 Dynamixel motors. Table 1 lists the pros and cons for each option we considered.

Motor	Pros	Cons
HV-5932MG	Low Cost Available in high torque variants Simple mounting style Easy Communication Style	No Feedback Can only perform position control Unable to tune PID Non daisy-chainable
Dynamixel AX-12a	Mid range Cost Easy to mount Allows for daisy chaining Allows for tuned PID Allows for Position, Velocity and torque control	Low Torque Difficult Communication protocol Large body
Dynamixel XH430	High Torque Easy to mount Allows for daisy chaining Allows for tuned PID Allows for Position, Velocity and torque control	Expensive Difficult Communication protocol Large body

Table 1: Pros & Cons for each motor considered

4.1.4.2 Motor Decision

Ultimately we decided to use the Hobby Servo. While the Dynamixel servos provided an all in one solution to the sensor integration and feedback we were looking for in our motors, the price for the amount of power we needed was way out of our budget. As a result we needed to come up with an alternative solution for providing feedback to our system. While we could have just added external sensors to our motors and joints, we decided to modify the servos to suit our needs. This was also part of our decision to choose the hobby servos, because after some research and thinking we discovered that we could fit our own electronics board and housing on the servos to essentially turn them into the smart Dynamixel servos for a fraction of the cost. The design of the servo electronics board is covered in more depth in the electrical implementation section of this paper.

4.1.4.3 Elastics Testing

Once we had decided on a servo we had a rough idea of the size of the robot we were intending to create. Based on this conceptual size and weight and the size and power of our servos, we felt that there was a possibility that an elastic suspension system would be needed to allow the robot to function properly. This system would work by attaching a spring or bungee cord or other elastic mechanism to the joints of the cat. When the cat is under its own weight the elastics work to reduce the amount of torque seen by the motor. Because most elastic systems are dependent on the distance they are compressed or extended by the farther the limb is moved the more additional torque is applied. One thing to keep in mind is that when the robot has lifted up the leg and is no longer under its own weight the elastic systems are now working against the motor. This means that however much additional assistance is applied by the springs, it can not be greater than the max torque output of the motor. In order to get an idea of the spring constants available we purchased a length of 1/4 bungee cord to test. The test was done by suspending a length of bungee cord from the top of a table and hanging weights on it to determine how far it stretches. Given

that Hooke's law is F = k*x where k is the spring constant and x is a linear distance we can easily determine the value of the spring constant.

Figure 22: Elastics Testing

Test	Weight	Initial Distance	Final Distance	k
1	2 kg	13.0 cm	16.5 cm	.57 kg/cm
2	2.7 kg	13.0 cm	18.75 cm	.47 kg/cm

Table 2: Spring Test: Tested bungee cords to determine the spring constant value

Our tests weren't perfect but they gave us a rough idea of what to expect for the spring constant of our elastics. Using this info we figured we could make a rough controller for the system and refine it once we had better tests using the motors. However, we actually ended up not using elastics because the results of the motor testing proved that the motors would be powerful enough to move the robot.

4.1.4.4 Motor Testing

Once the servo was chosen, we tested the actual torque output of several motors. This was done to confirm the specifications given to us by the supplier as well as to see the difference between motors of the same model. Knowing the motors' limits is crucial when designing the quadruped. To test our servos we created a laser cut lever arm with holes spaced 1 cm apart. By hanging a 1kg weight on the holes with the lever parallel to the ground and moving incrementally outward we could determine the actual torque output of the motor. We tested both the stall torques and operational torques. The operational torque was determined by adding weight until the motor could no longer move the weight. The stall torque was determined by adding weight until the motor could not physically hold the weight up.

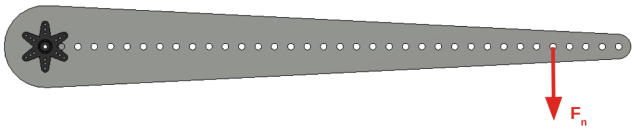


Figure 23: Graphic of how motors were tested

The HV-5932MG servos are advertised with a stall torques of 32 kg-cm. We found the servo was able to hold the 1kg weight at the 32 cm however it could not move it which is to be expected given that is its stall torque. From there we moved the weight closer in to determine at what point the servo could comfortably move the weight. We found that 20 kg-cm was the highest torque the servo could operate without any loss of function. Using this number as the known output torque of the motor will allow us to design a system that we are confident will be able to operate without falling down under its own weight.

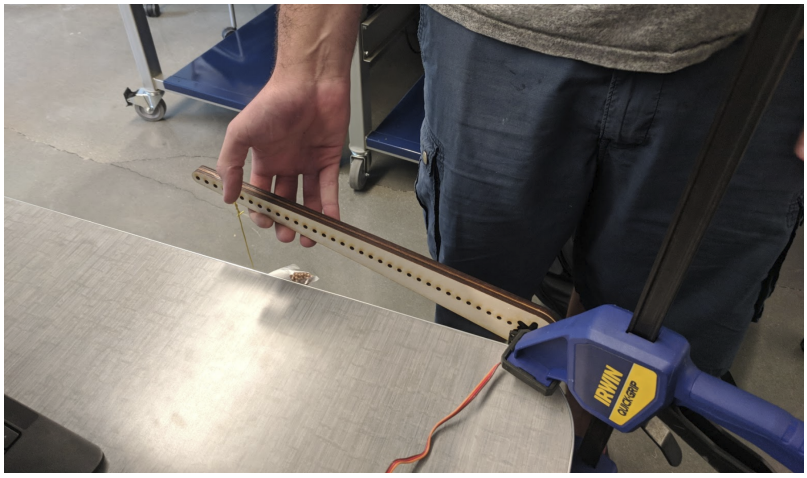


Figure 24: Testing the JX-Servo HV-5932MG

4.1.4.5 D-H and Forward Kinematics

With any robot there is math and analysis that needs to be done to make sure that the robot will properly do what you want it to. The majority of the math that was done for smallest was to ensure we had a proper kinematic representation of the robot so that we could control the robot as well as analysis it. As mentioned before kinematics provide us with a mathematical way to describe the position of the robot in space. This can be done by describing the lengths and angles of each joint and figuring out where in space the robot is (forward kinematics) or by giving a point in space and figuring out was the angles needed to reach that point are (inverse kinematics). The forward kinematics were solved by using the D-H convention as seen below.

Link	d	θ	r	α
1	0	θ_1	0	90
2	0	θ_2	l_1	0
3	0	θ_3	l_2	0
4	0	θ_4	l_3	0

Table 3: Table of DH parameters

$$X = a_3 * sin(\theta_2 + \theta_3) + a_2 * sin(\theta 2) + a_4 * sin(\theta_2 + \theta_3 + \theta_4)$$
$$Y = sin(\theta_1) * (a_1 + a_3 * cos(\theta_2 + \theta_3) + a_2 * cos(\theta_2) + a_4 * cos(\theta_2 + \theta_3 + \theta_4))$$
$$Z = -cos(\theta_1) * (a_1 + a_3 * cos(\theta_2 + \theta_3) + a_2 * cos(\theta_2) + a_4 * cos(\theta_2 + \theta_3 + \theta_4))$$

Figure 25: Forward Positional Kinematics

4.1.4.6 4 DOF Inverse Kinematics

The inverse kinematics were a bit trickier and required a little more algebra. In order to solve this problem we had to apply everything we knew about inverse kinematics for a 3 degree of freedom leg and apply it to a 4 degree of freedom leg. With a redundant 4 DoF system like ours, the leg has an additional limb inline with several others causing infinite solutions if you are just trying to solve the kinematics for an X,Y,Z point in space. As a result an additional term is added in to define the angle of the fourth link giving the system at most two solutions for a given x,y,z and θ value. Looking at the picture below you can see how this was done. By giving the leg an x,y,z value and an angle you can use some simple trigonometry to solve for an x,y,z the describes the 3 DOF system which we already know how to solve for. This is the most basic way to solve for this system. The math can get a bit more complicated when you start changing the reference point of theta 4. For example the method about describes theta 4 with reference to the other angles in the plane and is not changed if theta 1 moves. However if you describe theta 4 with respect to the ground its value is going to change once you start moving theta 1. We ended up working through both methods and implemented the second method so that we can define the angle with respect to the ground.

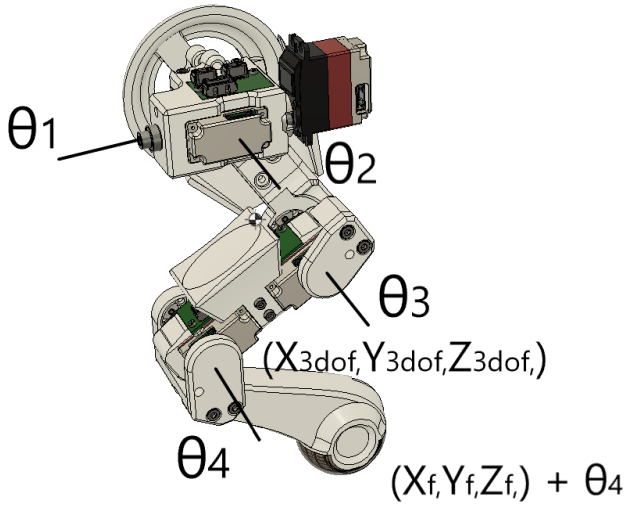


Figure 26: 4 DOF kinematics

4.1.4.7 Force Propagation

Once the forward and inverse kinematics had been solved for we were able to move forward to some more complicated analysis. Force propagation is a mathematical technique that allows us to simulate a force vector on the end-effector of our limb and figure out what the torque experienced by each limb is going to be. This is extremely useful as it allows us to test weather or not the robot is going to be able to hold its self up and actually move. To solve for the torques of the system a Jacobian matrix is created. This matrix is basically describing the relationship of the joint velocities and the end-effector velocity. It is created by taking the partial differential of the

forward kinematics with respect to each of the joints as seen below. Once the Jacobian is found determining the torques of the system is as easy as taking the transpose and multiply by the torque vector.

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Figure 27: Jacobian Matrix [9]

Force Propagation
2-Link Arm

$$\vec{\tau} = J(q)^T \vec{F}_{tip}$$

$$J(\vec{q})^T = \begin{bmatrix} -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & l_1 \dot{\theta}_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & 0 & 0 & 0 & 1 \\ -l_2 \sin(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} -l_1 \sin(\theta_1) - l_2 \sin(\theta_1 + \theta_2) & l_1 \dot{\theta}_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) & 0 & 0 & 0 & 1 \\ -l_2 \sin(\theta_1 + \theta_2) & l_2 \cos(\theta_1 + \theta_2) & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} f_x \\ f_y \\ f_z \\ n_x \\ n_y \\ n_z \end{bmatrix}$$

Figure 28: Force Propagation example with 2 DOF Arm picture courteous of our advisor Gregory Fischer

By changing several different parameters such as link lengths, weight (Force vector) and position of the leg we were able to get a good idea of what our design parameters were. This analysis allowed us to confidently move forward with the design of the robot. It is important to note that this is only an analysis of a static system. We did create a set of dynamic equations however those were not implemented into the force analysis. As a result we made sure to include a safety factor of around 2-3 to be sure that the system would be able to handle the additional torques cause by moving each of the links.

4.1.4.8 Tail Kinematics

The physical design of the tail is covered in the implementation section of this paper, however it was important to understand how the continuum tail would work and how we would control it before we made it. Continuum tails are a series of linkages that are controlled by pulling on a string running through the linkages, similar to how fingers work in a hand. Motion is determined by how much actuators pull on the three or more strings built into the linage. After some research we found a wonderful explanation of the kinematics for a simple continuum robot. The controls for the continuum were fairly simple as the tail was not the main focus of the project. The tail can actually be modeled as a flexible beam. There are three coordinate systems that are defined for this system. The base disk, end disk and bending plane coordinate system. The base disk coordinate frame is the reference system. The X axis points from the centre to the first secondary backbone, the Z axis is perpendicular to the disk, and the Y axis completes the frame according to the "right-hand rule". From there the bending plane coordinate system can be obtained by rotating the base disk coordinate around the Z axis for angle y. The last coordinate system is found from 4 steps. First, translating the centre O to Oe. Second rotating the local coordinate around its Z axis for angle γ. Third rotating the local coordinate around its Y axis for angle . And finally, rotating the local coordinate around its Z axis for angle -γ [23].

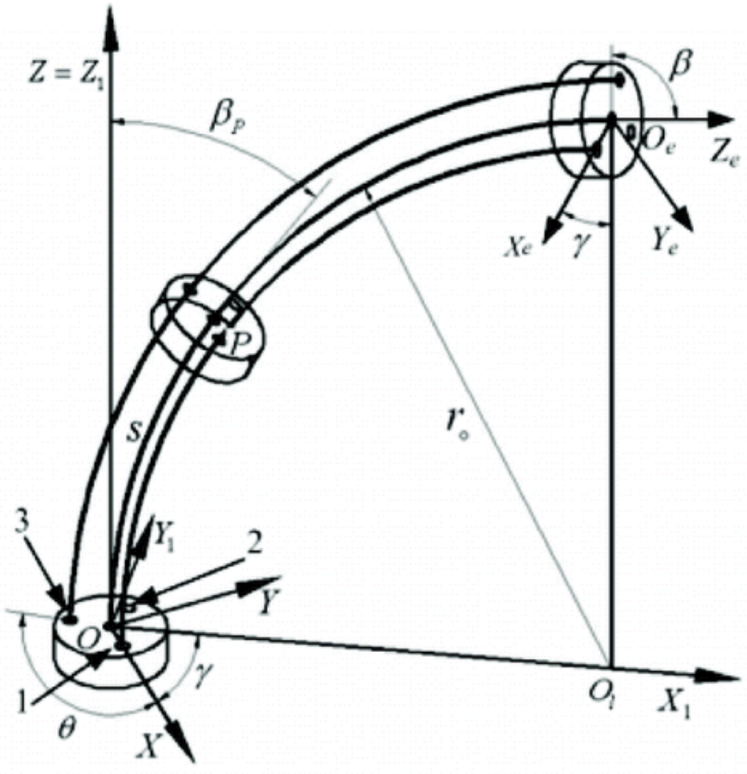


Figure 29: Frame setup for continuum tail [23]

Through some frame transformations and algebra we can determine the forward kinematics of the system. Because the segments are assumed to be equal and the system is uniform throughout You can determine the position of any point P on the tail by using the following equations [23].

$$P = [x, y, z] = [s/\beta_\rho(1 - cos\beta_\rho)cos(\gamma), s/\beta_\rho(1 - cos\beta_\rho)sin(\gamma), s/\beta_\rho sin(\beta_\rho)]$$

Figure 30: Continuum Forward Kinematics

- s = arc-length parameter of the segment OP (s=0 at the base disk and s=l at the end disk).
- β_ρ = the bending angle of the primary backbone tangent in the X_1Z_1 plane at the point P
- γ = the rotation angle of the bending plane.

As described before the motion of a continuum robot is controlled by changing the lengths of the driving wires. The change in length of each wire is controlled by motor and the motion of each motor is directly proportional to the displacement of the wires. The transformation between the driving space (motor movement) and joint space (angles given to the robot) can be expressed by the equations below [23]. Once you are able to determine the length of each of the driving wires it is a simple enough process to convert that to a specific number of rotations for each motor given the size of the cam used to spool the wire. This is how we controlled our continuum tail.

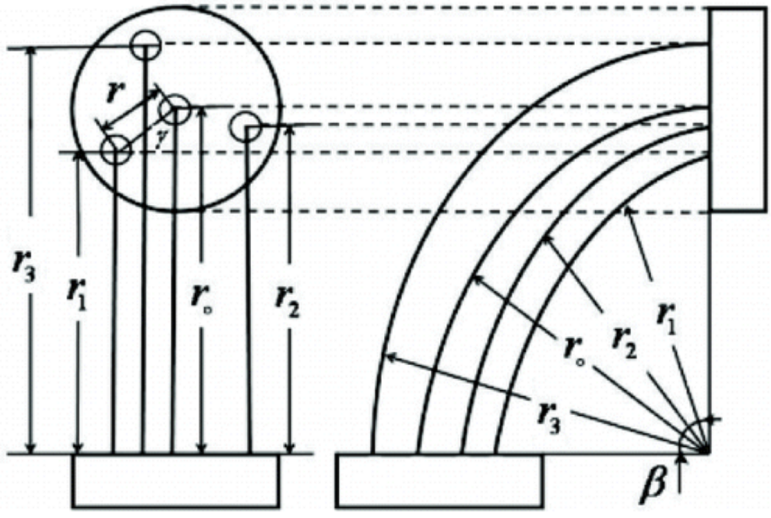


Figure 31: Drive wires for a continuum model [23]

$$q_1 = [r\beta cos(\gamma)]$$
$$q_2 = [r\beta cos(-\gamma + \theta)]$$
$$q_3 = [r\beta cos(\gamma + \theta)]$$

- ☐ q1,q2,q3 = the length of each driving wire.
- ☐ β = the bending angle at the end disk.
- ☐ γ = the rotation angle of the bending plane.
- ☐ r = the distance from the primary backbone to each secondary backbone on the disk.
- ☐ θ = division angle ($\theta = 2\pi$ / n, is the number of secondary backbones).

Figure 32: Continuum Forward Kinematics

4.2 Electronics & Low-Level Software

4.2.1 Requirements

- ☐ Low latency communication
- ☐ Feedback on position and torque from servo
- ☐ Minimal wires to manage
- ☐ Tunable PID constants
- ☐ Custom Foot Pressure Sensors
- ☐ High Speed Micro Controller
- ☐ Low latency, accurate IMU

4.2.2 Motor Controllers

After comparing each motor in Table 1, the solution arrived upon was to make our own hybrid motor, using the body and motor and gearbox of the HV-5932MG and implementing a custom motor controller board. This was done to allow for position, velocity and torque control to be done directly on the Servo as well as to allow for tunable PID constants, and feedback to the master controller. This custom servo motor will communicate is a custom written daisy chained SPI protocol. These custom motor controllers will have a STM32 micro controller and an MA702 absolute magnetic encoder[13].

4.2.2.1 Frequency

When designing the motor controllers 2 hardware timers were used for PWM pins which would allow for up to 4MHz PWM signals to be used. In the preliminary stages of testing an arbitrary value was chosen which equated to 1MHz switching frequency. When testing the motor controllers an issue that occured, mainly while testing positional control. When given a set point the motor would only actuate given a duty cycle of 50% this led us to testing a range of frequencies between 10kHz and 2Mhz. We found that a frequency of 20kHz allowed for a smooth motion of the motor without an un-pleasant sound.

4.2.2.2 Testing

The custom motor controllers went through multiple stages of testing and development. starting with the designing of the preliminary version of the board. When doing this the pre-existing physical locations had to be taken into consideration, these include the center of the output shaft, the dimensions of the servo housing and the location of the motor itself. These locations would be the determining external dimensions of the board, the location of the MA702 absolute magnetic encoder and the mounting location being directly to the power connector of the motor. After populating and assembling this version of the board minor errors were discovered with a foot print and other minor details. These issues were fixed and the boards reordered. When the updated version was received, it was tested and independently the motors worked. When daisy chained the effect of SPI cross talk became eminent. The chaining of MISO and MOSI would result in the encoder input of a 0x00 or null byte to pull the output of the next motors encoder low and vice-versa. This problem was solved by using 2 dedicated SPI channels, one for communication between motors and the other to be used for intra-motor communication.

4.2.3 Micro-Controllers

The project will require many micro controllers, each motor, foot sensor and IMU will require its own independent micro controller to communicate back to a master micro controller. Due to this two specific micro controllers were chosen, the STM32H743iit[20] and the STM32L432kb[21]. The STM32H732iit was chosen for its large amount of flash memory at 1Mb, six dedicated hardware SPI channels, high clock speed of 400 MHZ being the fastest low cost micro controller easily available and its ability to emulate a USB HID device for low latency communication between the micro controller and a computer. The STM32L432kb was chosen due to its low cost and high clock speed of 80 Mhz. The STM32L432kb is used in each motor to perform 3 PID controllers and drive each motor, each foot sensor to collect all pressure sensor data and report back to the master in order to remove the wait time between the reading of each sensor and each IMU to collect all the gyroscope and acceleration data and report it to the master controller in order to reduce the time taken to read the IMU.

4.2.4 IMU

When deciding on the IMU to use for this project there was a lot of consideration taken to the specific model chosen, IMUs from ST-Microelectronics, Bosch Sensortec and many other were considered but primarily the LSM303CTR, LSM6DS3USTR, BMX055, BMI160 and the BNO055[18] were considered. The BNO055 was chosen due to the great deal of available support available for the sensor, the on board fusing of the 3 major sensors on board and the previous experience the group

has had with the sensor. The BNO055 fuses the gyroscope, magnetometer and accelerometer in order to stabilize the measurements being read.

4.2.5 Foot Pressure Sensors

There was a large amount of existing research for these foot pressure sensors done by other groups. These include articles from Harvard[3][4], WPI[24] and MIT[22]. The research done by the group for the paper "Inexpensive and Easily Customized Tactile Array Sensors using MEMS Barometers Chips" [3] proved to be the most relevant when it cam to understanding the affect the encapsulation of the sensors would have. Due to their documentation using the MPL115A2 pressure sensor we were able to get a baseline for the optimal thickness of polyurethane to use for our project. We used the BMP280 barometric pressure sensor [17] as it had a much higher working range as well as greater sensitivity. The BMP280 pressure sensor allowed for a 24 bit pressure reading, the availability to use SPI to communicate with the sensor making it far easier to communicate with multiple sensors made it a simple choice.

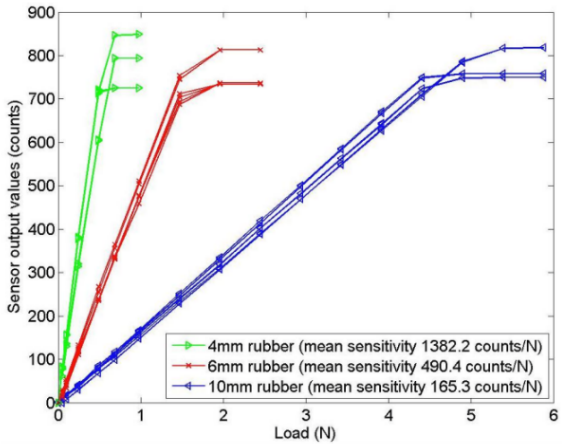


Figure 33: Correlation between load and pressure readings at different thicknesses of polyurethane[3]

The research done by the Contact Behavior of Soft Spherical Tactile Sensors[24] group demonstrated a similar method for calculating a pressure vector using magnets and hall effect sensors, despite the difference the math used to convert from readings to a pressure vector proved very useful to calculate out pressure vector using the 7 sensors on the foot of the robot.

4.2.6 Communication Protocol

4.2.6.1 Micro-controller to Micro-controller

The decision on how to communicate between the master micro controller and the multiple salves through out the system came down to ease of re usability. The major communication options included I^2c, SPI, Can Bus, RS485 serial and RS232 Serial. The following is a table giving pros and cons of each choice.

Protocol	Pros	Cons
I^2c	Simple 2 wire interface Easy Daisy chaining Available on most micro controllers Supports DMA	Requires independent addresses on each device Requires independent firmware to be flashed to each motor Non synchronous Requires motor to be re flashed to move to a different location
SPI	Synchronous Max baud rate of 16.5 Mbaud Supports DMA Available on most microcontrollers Daisy-chainable Allows for multiple slaves to share one chip select line therefore no addressing	Requires 4 wires to interface
Can Bus	Designed to be daisy chained Simple 2 Wire interface	Not available on most micro controllers Slow
RS485 Serial	Simple to use Designed to daisy chain	Requires external IC Slow Requires a software address be set
RS232 Serial	Available on most micro controllers Simple to use No hardware addresses needed	Slow non synchronous Difficult to daisy chain but possible

Table 4: Pros & Cons of each communication protocol

After researching in depth into each of these protocols taking into development time, limitation of resources such as space on circuit boards and ease of communication between micro controllers chosen, SPI was the final choice. The micro controllers used in the servo, foot pressure sensor and IMU breakout boards communicate back to the master micro controller through a daisy chained SPI DMA protocol where the packet of data is sent to the first object in the chain, the reply from the previous cycle is passed on as a return packet to the next, and so on until the final packet reaches the final micro controller and returns the packet of results to the main micro controller. The flow of data for an example leg is shown in the figure below.

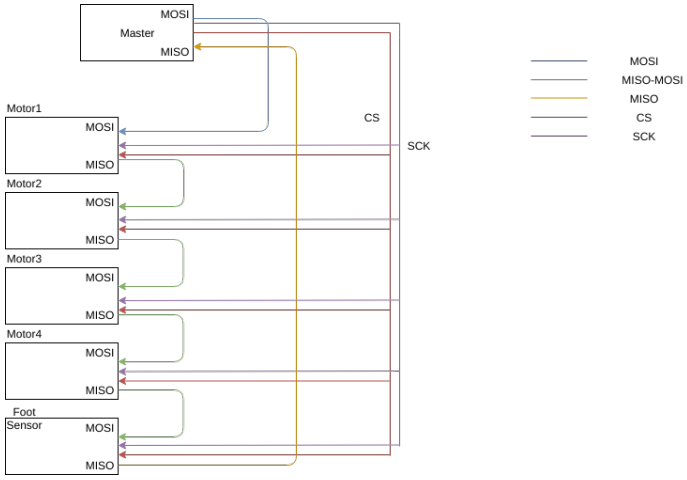


Figure 34: Flow of data through a leg

4.2.6.2 Micro-controller to Computer

There were many fewer options for communicating between a computer and the master micro controller. The major solutions included Ethernet, USB HID and UART over USB. These options each have their advantages and disadvantages. Ethernet, this requires external supporting circuitry to interface between the micro controller and computer, this combined with the priority level of the networking stack on any Debian distribution of Linux including Ubuntu eliminated this option as a viable solution. USART over USB was a viable option as the USB communication stack is marginally higher than the networking stack however USB to USART is limited to 2 Mbits/s. This meant that the throughput would not be enough for the robot to function reliably. USB HID communicates a 54 kbyte packet, reliably at 1ms. The HID USB stack is also the communication stack with the highest priority in any Debian based Linux system. therefore it is the least likely to lose a packet or have a data transmission error.

4.2.6.3 Processor cycles

The main purpose of the motherboard is to receive data from the computer over USB HID, parse, process and distribute to the motors and sensors accordingly. After speaking with Boston dynamics and deciding that for a reliable dynamics controller we would ideally have a control loop speed of 1kHz at the highest level. This means that in 1 ms the micro controller must receive 64 bytes of data from the on board computer, parse it to determine the type of command being sent to the motors, whether it is a position, velocity or torque set point, populate the buffer and send through SPI to each motor assuming all read and writes to memory and GPIO is atomic, the time for data transfer can be calculated.

$$\text{Clock Speed} = 384\text{MHz}$$
$$\text{Data Size} = 16\text{bits}$$
$$\text{SPI Transfer Speed} = 24\frac{\text{Mbits}}{\text{s}}$$
$$\text{Size of SPI Package} = 5$$
$$\text{No. Devices} = 5$$

Time per clock cycle

$$\frac{1}{\text{Clock Speed}}$$
$$\frac{1}{384000000}$$
$$= 2.5 * 10^{-9}\text{s}$$

No. bits per leg

$$\text{Data Size} * \text{Size of SPI Package} * \text{No. Devices}$$
$$= 16\frac{\text{bits}}{\text{packet}} * 5\frac{\text{packets}}{\text{device}} * 5_{\text{Devices}}$$
$$= 400\frac{\text{bits}}{\text{leg}}$$

No. Clock cycles per SPI bit

$$\frac{\text{Clock Speed}}{\text{SPI Transfer Speed}}$$
$$\frac{384_{\text{MHz}}}{24\frac{\text{MHz}}{\text{s}}}$$
$$= 16_{\text{Clock Cycles}}$$

Time to update a single leg

$$\text{No. Clock cycles per SPI bit} * \text{Time per clock cycle}$$
$$16 * 2.5 * 10^{-9}$$
$$= 4 * 10^{-8}\text{s}$$

Time to update all legs

$$\text{Time to update a single leg} * \text{No. Legs}$$
$$4 * 10^{-8} * 4$$
$$= 1.6 * 10^{-7}\text{s}$$

Figure 35: Calculations to determine number of clock cycles elapsed during SPI transmission

Using the same SPI channel and DMA, the total time to transfer all data to each of the legs would be 1.6*10^{-7}s, this will allow the USB packet to be received, transmitted to the motors and the updated data returned to to the computer via an updated USB packet.

Continuing with the assumption of atomic data transfers the use of separate SPI channels for each leg would allow for update speeds of 4*10^{-8}s minimizing the total time in between receiving and sending a return packet. This would be the optimal solution as it allows us to keep the 1ms round trip for USB to be kept as well as allowing for other processes to be run.

4.3 High Level Software

4.3.1 Requirements

☐ Easy development through simulation

☐ Able to run headless on a single-board computer

- ☐ Fast cycle times able with little overhead
- ☐ User interface capability to control the robot

4.3.2 Robot Kernel

The kernel's main job will be to run the kinematics and dynamics engine. This includes balancing, foot-step planning, and all gaits to be developed (see gait section). To speed up development time and decrease cycle-times in certain calculations, an existing platform should ideally be used. Several platforms were looked at when making the decision.

Bowler Kernel: The Final Decision

Bowler Kernel is a Java-based software platform that is designed to run with low latency on most machines. Bowler Studio, on the other hand, is a development platform that runs on top of Bowler Kernel, and offers a basic simulation suite and user control options. These two platforms were used on for software development on the previous SmallKats (V1 & V2). With its built-in kinematics engine and a basic walking gait developed during previous projects, it has a good set of tools that allow for us to do hit the goals we want. It also has a convention for defining robots in a .xml document, which contains dimensions and DH parameters of the real robot. One of its biggest downsides, however, is it's lack of documentation. Software needs to be developed by looking directly at source code for method calls and researching existing implementations for inspiration for a project.

We chose this for a few main reasons. First, it checks all the requirements we set out. We were especially impressed at it's speed on a Raspberry Pi 3, running dynamics and kinematics often in sub millisecond cycles. We also had software already developed, like a basic walking gait, which would shorten development time and get us to a functioning prototype faster. Bowler Studio & Kernel also implements a UDP communication stack in order to communicate between both connected microcontrollers and computers. This allowed us to modify existing software to control our robot. Finally, we had access to its creator: Kevin Harrington. This allowed us to develop despite the lack of documentation, since we could ask him for help if we ever need to

Robot Operating System (ROS)

ROS, or Robot Operating System, is a widely used open-source robotics platform designed for adaptability and ease-of-bring-up for new robotic platforms. There are currently two major versions of ROS: ROS 1 and ROS 2.

ROS 1 has been tried and tested in the real world for over a decade. It is written in C and C++ and uses TCP/IP to communicate between processes, or nodes. ROS 1 is a really powerful tool for research and development, due to its adaptability and customizability. ROS's biggest advantage is the shear amount of packages developed for it. From camera software to SLAM algorithms to kinematics engines, ROS 1's community adoption along its open-source nature has led to a surplus of open-source packages that makes development easier. However, ROS 1 has two major downfalls: its speed and its security. Since ROS 1 is built using a networking back-end, the communication speeds between nodes is slow and unreliable. Testing proved that the guaranteed real-time performance for ROS 1 is about 100ms, about 100x slower than what we need for our system. It also relies on all ports to be open for good communication, making it a security nightmare. Most robotic solutions are initially developed in ROS for quick development time, but

are then ported to a custom platform once key algorithms and functions are developed. Other research and development platforms, similar to ours, use ROS as a user interface and testing suite, with custom software written to communicate with ROS and the hardware.

ROS 2 is an attempt to fix the multitude of problems with ROS 1. It uses UDP sockets instead of the TCP/IP protocol for faster communication between nodes. It also implements new security protocols as a part of the platform. It is also built in a way where any language can be used, because its core is easily wrapped by any modern language. However, ROS 2 still has its issues. For starters, ROS 2 is relatively new: just a few years old in a semi-stable form. It is not very feature-rich, and is not backwards compatible with ROS 1 packages (there is a backwards compatibility tool, but is very buggy at the time of writing). The second big problem is the instability of speed. ROS 2 boasts that it has been built to be real-time. However testing this claim, we found that although it could guarantee a 1ms loop 95% of the time, the cycle time varied greatly - from 10 microseconds to 2.7 milliseconds. Although ROS 2 may be very viable in this situation, we determined it's relatively young life as well as its low adoption in the industry made it impracticable for use.

IHMC Open Robotics Software

The IHMC Open Robotics Software is a library set developed by Florida Institute for Human and Machine Cognition (IHMC) designed to make it easier to develop bipedal and quadrupedal walking platforms. It includes a physics simulator and a kinematics calculator written in Java that will allow speedy calculations with close to real-time performance, while still offering a simple and robust development platform. The IHMC library can be split into two parts: the kinematics engine and the simulator.

The kinematics engine built into the IHMC library uses the Euclid Core library (also developed by IHMC) to calculate matrices and perform rigid body transformations. Since this library is designed in Java, the main concern is its performance in a real-time environment. However, if designed properly, Java code can actually be faster on subsequent runs than a C++ or C application can be. This is because the Java run time Environment (also known as the Java Virtual Machine, or JVM) can optimize certain parts of the code by making assumptions after the first instance has run. This is true as long as the developer is careful to avoid using the garbage collector built into the JVM, since it is very computationally intensive. Java also makes development easier, since Gradle can be used to manage and pull together libraries, when compiling into an executable. Finally, Java applications are cross-platform, so one executable can be run on any device with a JVM. This makes deployment on several different robots easy, since executables can be compiled locally before pushed to other robots running different processors.

The IHMC libraries also include a simulator designed to prototype software. It allows to simulates any developed code at its core instead of risking hardware. The simulator is also better integrated with the other libraries to be used, since it is developed by the same group.

When tested, we found this platform to have many powerful tools that would shorten development time significantly. However, the biggest issue that kept coming up was its lack of documentation, since it only had recently been made open source. The IHMC libraries are certainly a good decision for anyone developing a walking robot in the future after the research organization develops and releases better documentation for its platform.

Custom Platform

Since neither ROS nor Bowler Studio is exactly what we were looking for, a custom platform could have been written. It would be likely written in C and C++ and communicate through processes

using a standard inter-process communication protocol, or IPC. This would allow for our platform to be extremely fast, limited only by the Linux kernel and processor speed, while maintaining all communication open locally. It can also be easily adapted to send data over other protocols, like SPI, USB, or TCP/IP or UDP Sockets. However, developing a custom platform is a project in itself, and adds a lot of development time and work to ensure proper stability and functionality. As fun as it sounded, we determined it was outside the scope of our project.

4.3.3 Robot User Interface

Since Bowler Kernel was chosen as the main development platform, the user interface was a relatively straight forward decision: Bowler Studio with a controller for wireless communication. Bowler Studio allows for the user to control the robot being developed when plugged in to the development computer. It also gives the developer a virtual view of the robot, like joint angle positions and robot states. We also used a wireless controller to control the robot when running on the Raspberry Pi, since there was no way to effectively control the robot wirelessly through Bowler Studio.

5 Development and Implementation

5.1 Mechanical

Based on our research and testing we decided to make advancements in what is considered the typical quadrupedal design. We set out to creating a system that more closely resembles a cat by implementing 4 DOF legs and a continuum tail. Our electronic systems will include pressure sensing feet with multiple sensors to allow for the triangulation of the pressure vectors, an adaptive number of IMUs which can be placed in optimal locations and custom smart servo motors capable of performing constant position, velocity and torque. We will be creating custom control systems that will allow us to implement a variety of gaits. We hope to be able to at least create a basic walking and running gait and experiment with jumping.

5.1.1 Tail Design

When looking at other quadrupeds, tails are often not used, as they add complexity in the design. However, we saw a tail as an opportunity to better control our center of mass of the robot. This would give the user more body control, and could be removed if needed. We ultimately looked at two different designs for the tail: a solid tail and a continuum tail. A solid tail has articulation at the base of the tail, and is relatively simple to control. All normal kinematics and dynamics apply to model the movement of the tail, and modeling the system is relatively straight forward. Users can use a solid tail to move a mass left and right to control the center of mass of the robot as a whole. Modeling continuum tails is significantly more difficult, since every linkage has to be modeled as its own joint and link, complicating the system. However, users have some more control over the center of mass of the tail, especially if you stack continuum manipulators on top of one another. In order to give users an option, both a continuum tail and a solid tail were developed, and can be swapped out to simulate different situations. Figure 36 shows the difference between a solid and a continuum tail.

Figure 36: SmallKat's Solid Tail [Left] vs Continuum Tail [Right]

5.1.1.1 PLA and NinjaFlex Experimentation

The design for a continuum robot is pretty straight forward. There are a series of disks that are driven by 3 or more wires. Several designs will include springs or means of tension to provide some resistance and stability to the system. These springs will either be placed along the backbone of the tail or in between each disk along the path of the springs. When brainstorming ideas for how we were going to make the tail a thought came to us. By 3D printing the tail out of a firm flexible material we could customize the design and make the assembly process a lot easier. There were several iterations of this design made. The first tail was printed entirely out of NinjaFlex. The design provided the firmness and flexibility we were looking for however the design wasn't suited for bending as much as we needed it to. Additionally NinjaFlex prints much slower than other filaments. As a result we decided to combine our 3D printed idea with other standard designs that we researched a create a design with a single cored backbone. In order to cut down on print time we ended decided to experiment with dual extrusion printing and use two materials to create our tail. The result was several iterations of PLA and NinjaFlex tails that had different sized plates and cores. The final design resulted in a fair amount of success and was actually turned into a fully functional prototype.

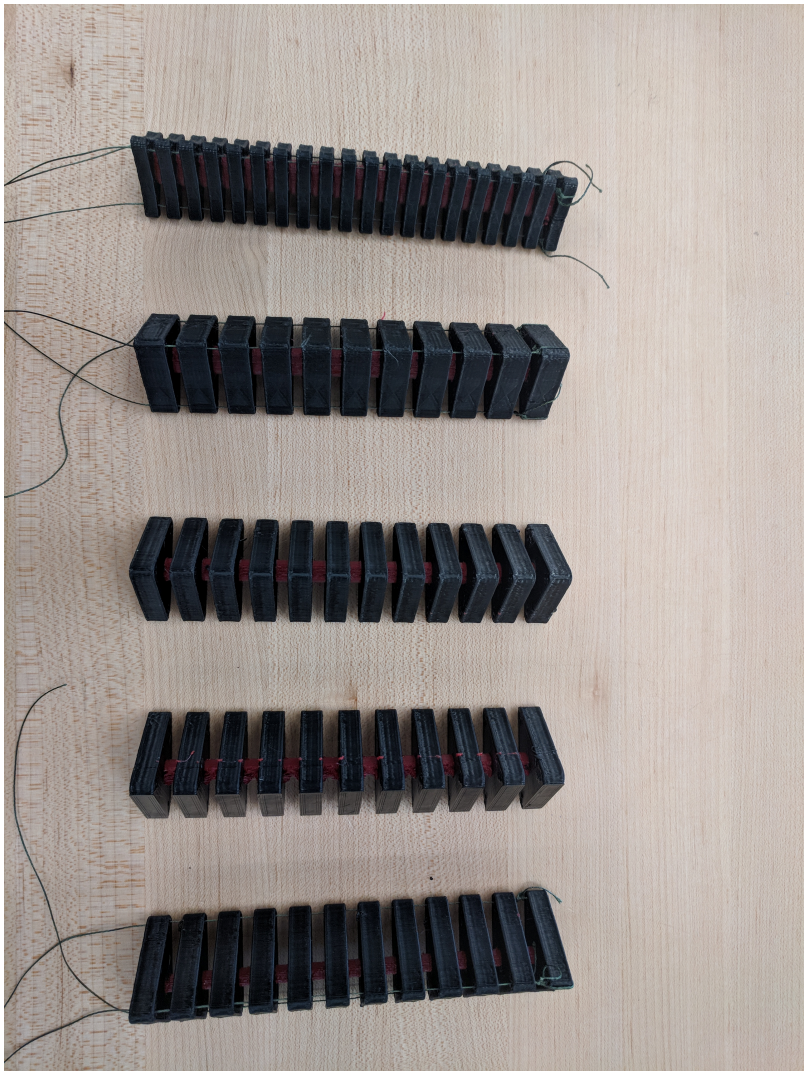


Figure 37: Dual extruded designs

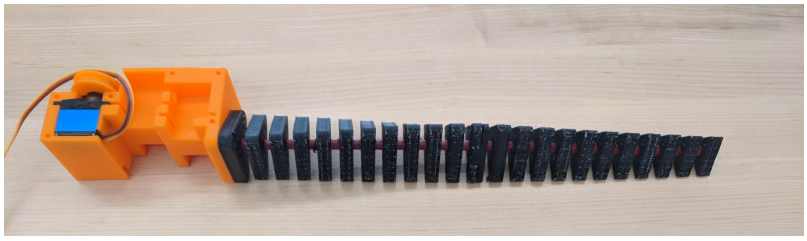


Figure 38: First full prototype of NinjaFlex tail

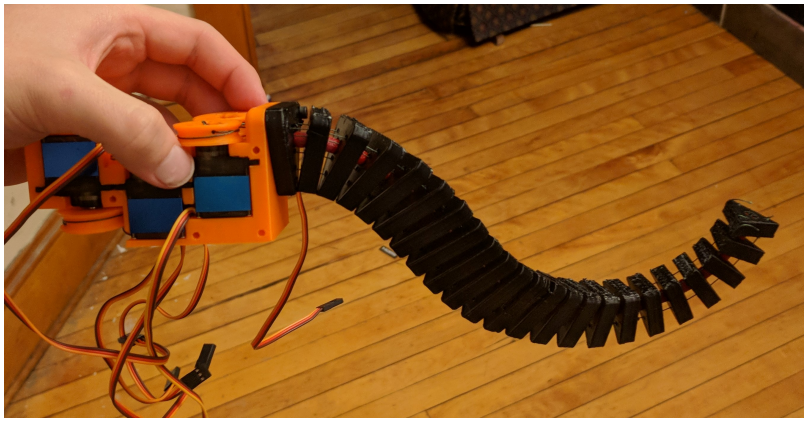


Figure 39: Prototype of NinjaFlex tail with strings

5.1.1.2 Universal Joint Design

While the dual extruded design surpassed our expectations it didn't quite provide the level of performance we were looking for. In our tests we were able to control the tail and it was flexible enough, however there was to much play in the system. There was nothing preventing the tail from waving widely out of control when bumped or moved. There is a possibility that this could be fixed with several more tests and iterations of the design however we decided to move to a more tried and true design. The next design utilized a universal joint that would allow the system to maintain a little more rigidity than the previous design. Each of the segments would be 3D printed, which also allowed us to change the size if needed and the whole thing was held together with small dowel pins and M2 bolts. The design can be seen below.

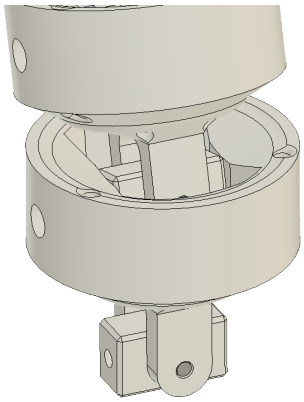


Figure 40: Universal Joint

Figure 41: Universal Joint Tail

5.1.1.3 Final tail Design

The tail design that ended up on the robot had two segments that could be individually controlled. Each segment was controlled by 2 Maxon motors that would spool the top two strings of the three. The last string was connected to a spring that would a apply constant force to the tail causing it to curl down. The rational behind this was that the tail would only need to move left right or up and these motions could be done with only two motors. Additionally we saved a lot of space by eliminating two motors. The motors had encoders on them which allowed us to track their position. However they were not absolute meaning that once the robot was turned off they would lose their position. To solve this problem a set of limit switches were added in so that the motors could move until they triggered the limit switch and home their position. The final motor setup can be seen below.

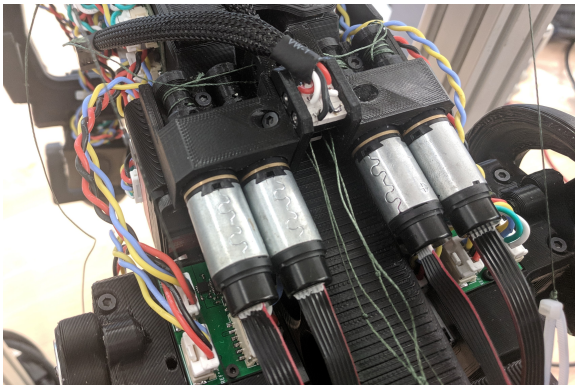


Figure 42: Final tail design

5.1.2 Design Iterations

5.1.2.1 Sizing SmallKat

When deciding the size of SmallKat, we looked into the options currently provided on the market, as well as previous versions (talked about in Section 3.1). Larger robots like Boston Dynamics' Big Dog and Spot Mini robots were rather large, and would increase the cost of the robot as a whole. When looking at SmallKat V2, we appreciated its "desktop" size, but realized it's small form factor also limited us in our goals. Since deciding on an exact size was difficult, we decided to size our robot based on a standard servo. This would allow users to swap motors with others that better suit their needs without having to redesign the whole robot. In the end, our robot ended up being slightly larger than a standard house cat. While the robot is still small and relatively nimble, it also has the capability to have advanced sensors, like a 3D Real Sense camera, and to traverse paths that the smaller SmallKats couldn't, like stairs.

5.1.2.2 First Design

The first leg design of the SmallKat MQP design was just a rough draft without specifications of motor torque or how big the leg could be. The design experimented with a design for the elastic spring system that used an embedded spring instead of the bungee cords we had tested with because it was designed before those tests.However, there was some difficulty embedding a big enough spring. Because there weren't many technical specifications to go off this iteration focused on aesthetics and making a design that could be printed.

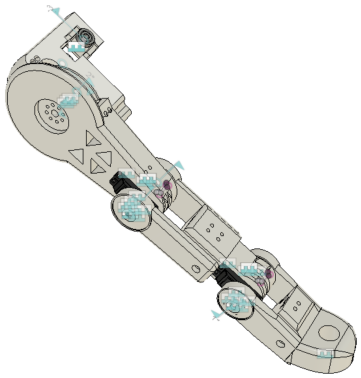


Figure 43: First Prototype Leg

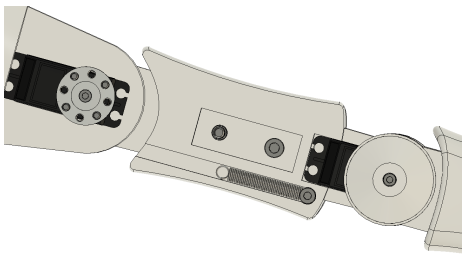


Figure 44: Integrated Spring Design

5.1.2.3 Second Design

The second design was also made to be the largest possible leg with the motor torques and the spring assist. We were able to do this by incorporating the bungee cords that were tested earlier in the project and the motor torque data that we had collected. In order to attache the bungee cords the were stuck through holes in each limb and then set screwed into place. Channels then guided them around the leg allowing them stretch when the leg rotated. We ended up printing and assembling this design however, it turned out to be very large. After further deliberation it was

decided that if we could make the robot smaller and not have to need a elastic spring system it would a lot easier to control in the control system.

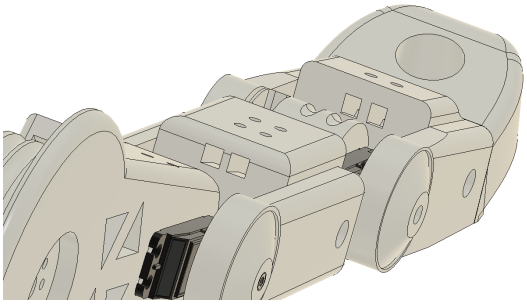


Figure 45: Second Prototype Leg

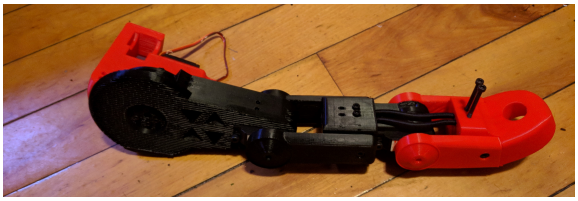


Figure 46: Second Prototype Leg Printed

5.1.2.4 Third and Fourth Iterations

The third design experimented with a new way of mounting servos. The two designs before this had the servos in the middle of the leg. However this design has the support coming from the opposite side of the servo horn and allows for the servo to stick out. This design was the first to be iterated with a mock up of the full body. It wasn't a full mock up but the other components were placed in to give a feeling for size and shape. It was determined that in its current configuration the servos on the inside would interfere too much with each other and cause the legs to not be able to move in all the way. As a result the fourth design was modified to have the servos stick out on the outside. This design was the first to be completely printed. The servos were turned to face outwards allowing for mobility on the inside and the legs shorted from previous designs so that it could stand under its own weight without the need of elastics. The overall design was very wide and a bit heavy. Many of the 3d printed parts were over engineered and didn't need to be as thick as they were.

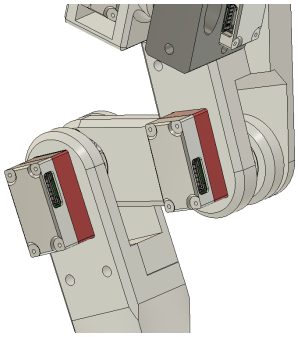


Figure 47: Side view of third design

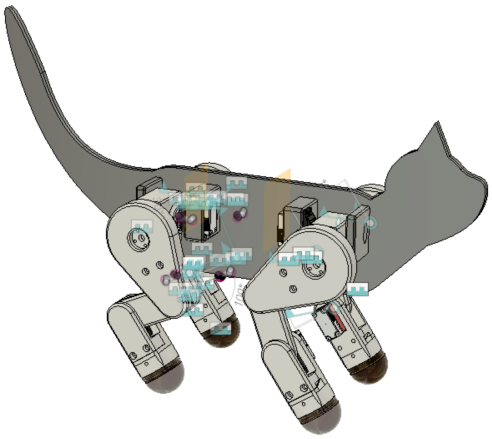


Figure 48: Prototype 3 with body outline

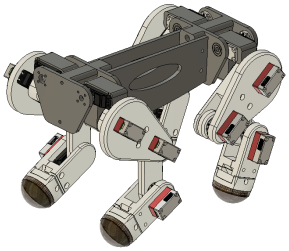


Figure 49: Full CAD Model of Prototype 4

Figure 50: Printed Version of Prototype 4

5.1.2.5 The Conceptual Fifth Design

This design was an idea that played off some of the same design feature as the previous design, mainly the fact that the servos were placed on the outside to allow for more mobility and thinner legs. The mid-leg was designed to be machined out of aluminum with the intention of it being much smaller and lighter than previous designs. The top shoulder joint was made significantly smaller than previous designs. The concept was never finished as the supports for the top leg were never added and the legs were still wider than desired. It did however provide some insight into the final design.

Figure 51: Prototype Design 5

5.1.2.6 First Full Design

After all the revisions and designs, we took everything we had learned and created the first full design of the SmallKat MQP project. The leg design was reverted back to having the servos in line with the legs. By simplifying and reducing the size of the 3d printed part of the legs we were able to keep the size of the hips to a reasonable size relative to the rest of the body. The body itself was designed in two parts for the front and back hips where the legs connected. The tail was a static design with 2 degrees of freedom as the continuum tail had not yet been finished. The head was the same model that had been used for the smaller previous versions of SmallKat except that it was scaled up inside. The neck of this design was given 3 degrees of freedom. The hope was to allow for more movement and give the cat a more natural feel. However it became to complicated and the neck was reduced back to 2 degrees of freedom in the final version. The center of the body held the custom battery back that was made for the robot. This utilized a trap door on the underbelly of the cat that allowed the user to remove the battery when needed. This design was fully printed and assembled to allow us to see size and connections. Below are some pictures of the fully completed prototype.

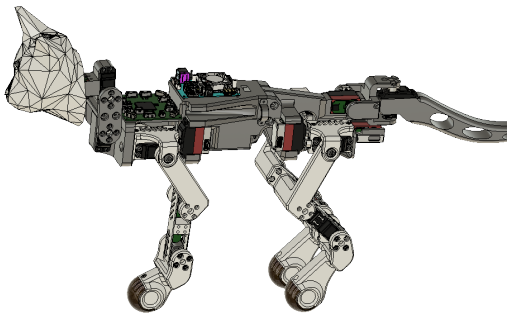


Figure 52: CAD of Prototype 6

Figure 53: All four SmallKats posing for a picture

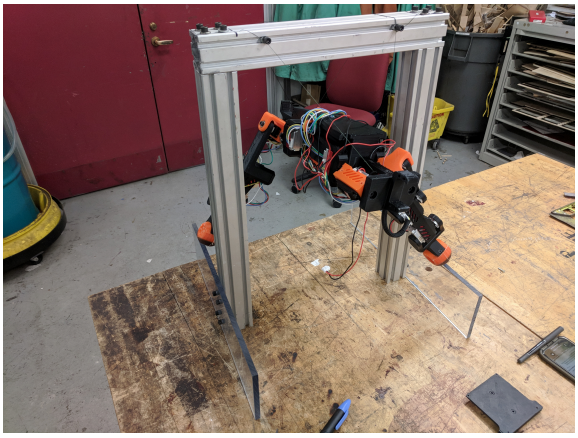


Figure 54: Prototype 6 up on a custom test jig designed to support the cat during testing

5.1.2.7 Final Design

Using everything we had learned from our testing, design iterations and prototypes we were able to create a final design. This design was very similar to Prototype 6 with a few changes to the body and the leg design. Additionally it incorporated the continuum tail and had a few additional aesthetic pieces. Overall we were very pleased with the final mechanical design. It was sleek, functional, compact and looked good.

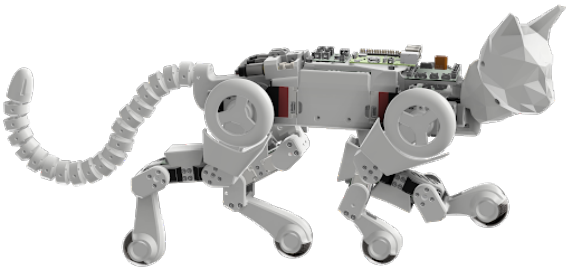


Figure 55: Rendering of the Final Design

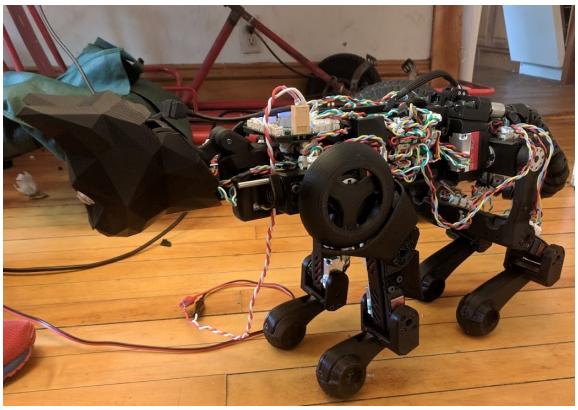


Figure 56: Fully assembled final design standing powered under own weight

5.2 Electrical

5.2.1 Low-Level Control Software

The low level control software of the robot can be broken down into four distinct subcategories. These include the master controller, foot sensors, motors and IMUs.

5.2.2 Communication Structure

When the motherboard receives trajectory points from the computer over USB HID, the controller then parses the data received to each motor. This controller performs calculations in order to optimize leg trajectory and paths. The data is packaged into four byte packets shown below.

Device ID	Command	Val 1	Val 2	Val 3

Figure 57: Data packet structure

The data is then sent to the motors, foot sensors and any other devices connected in the chain. Due to the nature of the system the data must be sent in order of the last device first as shown below.

Foot Sensor	Motor 4	Motor 3	Motor 2	Motor 1

Figure 58: Order of packets sent to a leg

The path of data follows the structure below.

Packet	Action of data
Packet 1	• Motor 1 Receives the Foot Sensor packet, replies previous position, velocity and torque to motor 2 • Motor 2 Receives return data from Motor 1 and replies its data to Motor 3 • Motor 3 Receives return data from Motor 2 and replies its data to Motor 4 • Motor 4 Receives return data from Motor 3 and replies its data to the foot sensor • The foot Sensor Receives return data from Motor 4 and replies the readings from the pressure sensors to the master controller
Packet 2	• Motor 1 Receives the Motor 4 packet, replies the Foot Sensor packet to motor 2 • Motor 2 Receives the Foot Sensor packet and replies Motor 1 data to Motor 3 • Motor 3 Receives return data from Motor 1 and replies Motor 2 data to Motor 4 • Motor 4 Receives return data from Motor 2 and replies Motor 3 data to the foot sensor • The foot Sensor Receives return data from Motor 3 and replies Motor 4 data to the master controller
Packet 3	• Motor 1 Receives the Motor 3 packet, replies the Motor 4 packet to motor 2 • Motor 2 Receives the Motor 4 packet and replies the Foot Sensor packet to Motor 3 • Motor 3 Receives the Foot Sensor packet and replies Motor 1 data to Motor 4 • Motor 4 Receives return data from Motor 1 and replies Motor 2 data to the foot sensor • The foot Sensor Receives return data from Motor 2 and replies Motor 3 data to the master controller

Packet 4	• Motor 1 Receives the Motor 2 packet, replies the Motor 3 packet to motor 2 • Motor 2 Receives the Motor 3 packet and replies the Motor 4 packet to Motor 3 • Motor 3 Receives the Motor 4 packet and replies Motor 1 data to Motor 4 • Motor 4 Receives the Foot Sensor packet and replies Motor 1 data to the foot sensor • The foot Sensor Receives return data from Motor 1 and replies Motor 2 data to the master controller
Packet 5	• Motor 1 Receives the Motor 1 packet, replies the Motor 2 packet to motor 2 • Motor 2 Receives the Motor 2 packet and replies the Motor 3 packet to Motor 3 • Motor 3 Receives the Motor 3 packet and replies Motor 4 packet to Motor 4 • Motor 4 Receives the Foot Sensor packet and replies Motor 1 data to the foot sensor • The foot Sensor receives Foot Sensor packet and replies Motor 1 data to the master controller

Once the cycle is completed the motors update the PID controllers, the foot sensors collect the pressure data and the cycle is looped. The return packet order is shown below.

Motor 1	Motor 2	Motor 3	Motor 4	Foot Sensor

Figure 59: Order of Data returned from a leg

5.2.2.1 Motor Controllers

When the motor receives a packet it checks if the first byte if the packet and compares it to the device ID of the motor. If the device Id does not match the first byte of the packet it is stages for transmission on the next packet received. If the packet is used to update that motor, the second byte of the packet is checked and compared to different known options.

Value	Command
0x22	Update device ID based on byte 3 of the packet
0x47	Updates Position PID constants based on bytes 3,4,5 of the packet
0x48	Updates Velocity PID constants based on bytes 3,4,5 of the packet
0x49	Updates Torque PID constants based on bytes 3,4,5 of the packet
0x91	Updated position set point in Position PID loop
0x92	Updated position set point in Velocity PID loop
0x93	Updated position set point in Torque PID loop

Table 6: Command values for motors

In between receiving packets the motors run multiple PID loops simultaneously to control position, torque and velocity at greater than a 10kHz refresh rate.

5.2.2.2 IMU

The IMU micro controller continuously samples the BNO055, recording the current gyroscope, acceleration and magnetometer data. On request from the master controller the IMU checks byte 1 and compares it to the device ID, if it matches it compares byte 2 to the list of known commands and replies accordingly which can be see below.

Value	Command
0x22	Update device ID based on byte 3 of the packet
0x37	get gyroscope data
0x38	get accelerometer data
0x39	get magnetometer data

Table 7: Command values for IMU

5.2.2.3 Foot Sensor

Similarly to the IMU, the foot sensor micro controller continuously samples the 7 pressure sensors. On request from the master controller the foot sensor checks byte 1 and compares it to the device ID, if it matches it compares byte 2 to the list of known commands and replies accordingly which can be see below.

Value	Command
0x22	Update device ID based on byte 3 of the packet
0x32	return most recent foot pressure sensor readings

Table 8: Command values for foot sensor

5.2.3 Motors

The custom servo motors posed a very interesting challenge when it came to size and component density. On a board that had to fit in an 18mmx36mm space a micro controller, magnetic encoder, voltage regulator, 2 connectors, a motor driver and a motor along with all supporting circuitry.

This along with some simple oversights meant the board went through 2 major revisions and 1 minor revision.

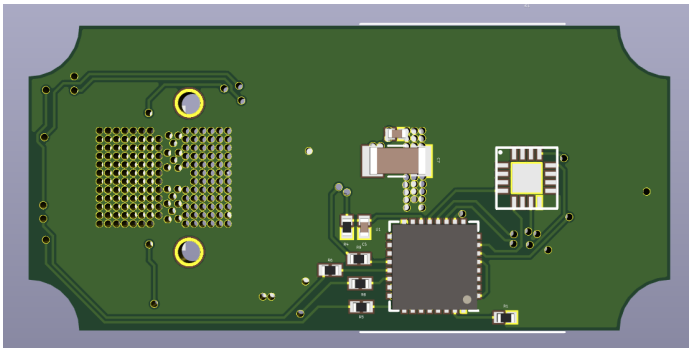


Figure 60: Motor controller PCB bottom

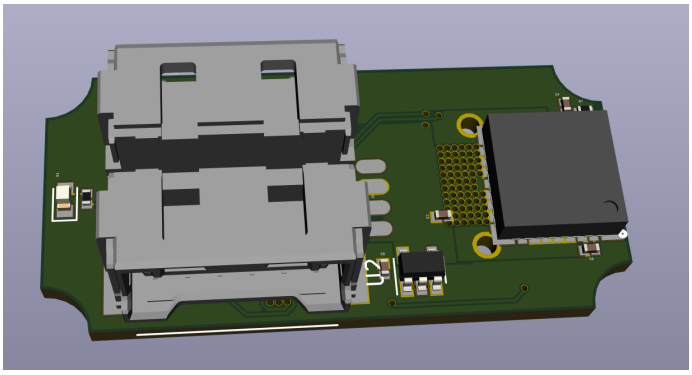


Figure 61: Motor controller PCB top

The boards were designed to be assembled directly into the motor connected directly by the motor pins, aligning the encoder above the magnet connected directly to the output shaft of the motor. The programming header was designed as a series of pads that can be used in combination with a Pogo pin programming jig, meaning that the firmware can be updated quickly and easily.

5.2.4 Motherboard

The mother board was developed to be a stand alone micro controller using the STM32H743vit6 micro controller running at 400 MHz. Broken out from this micro controller were all 6 SPI channels, 12 GPIO pins, UART, USB HS and a programming header. In order to limit the number of traces interfering with the High power coming from the battery, 3.3v and the Ground Plane a 4 layer board was chosen despite the increased cost of production. This meant that all the signal traces could be run internally allowing for a continuous trace running around the periphery of the board connecting all of the connectors to the 8.4v supply coming in from the battery.

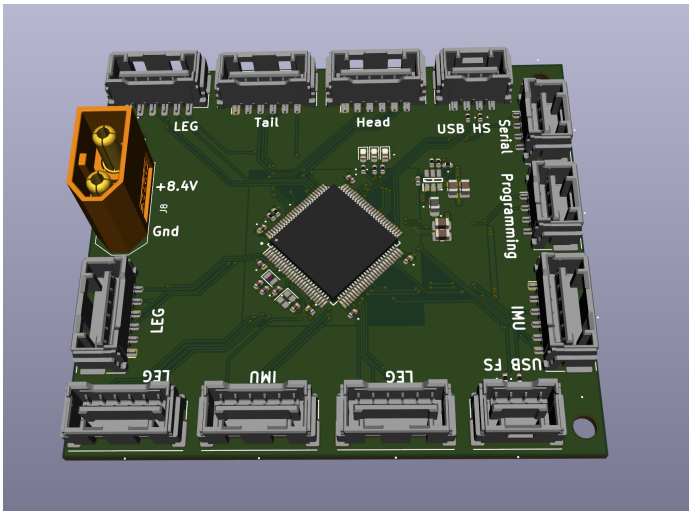


Figure 62: Motherboard

To ensure reliability at higher clock speeds both a 32.768 kHz crystal oscillator and a 32MHz crystal oscillator were used. While developing the firmware for the motherboard the the clock speed was reduced from 400MHZ to 384 MHz in order to optimize the speed at which the SPI could be run as a clock divisible by 24MHz was requires to take full advantage of the SPI communication with the micro controllers on the motors. In order to have a reliable and stable 3.3v rail for the micro controller, 2 switch mode buck converter power supplies were used in order to stop regulate the 5v supply from USB and the 8.4v supply from the battery pack. This allows the board to be developed on and tested using only power from the USB connection and does not require the battery be connected.

The motherboard was designed and went through one minor and one major, the minor revision was done to remedy an error in the switch mode power supply circuit as well as to change from USB FS to USB HS allowing for a much faster data through put at up to 480Mb/s. The Major revision was done in order to switch micro controllers to a micro controller package much better suited for the application. Instead of using the 176 pin package of micro controller, the 100 pin variant was capable of performing the same tasks required. In addition to the change in micro controller, Both available USB ports were broken out and much more safe guarding against inductive spikes affecting the master micro controller were implemented due to the death of 2 of the previous revision. Though many of these changes were very beneficial, the changing of microcontrollers came with a single draw back, the loss of 1 SPI channel, meaning some motors were forces to share SPI channels with other devices on the robot. This prevented us offloading the processing of SPI data to the hardware DMA as it had the possibility of cross talk between multiple devices. The breaking out of both USB ports did allow us to take advantage of the processing power and speed of the micro controller. By combining both ports we were effectively able to double our throughput, sending data and requests for different devices through different ports. Using the USB HS ports for motor data and the USB FS port primarily for IMU data requests allowed us to remove the bottle neck of having to wait for the data transmission to be completed and allowed us to increase the sample rate of the IMU.

5.2.5 IMU

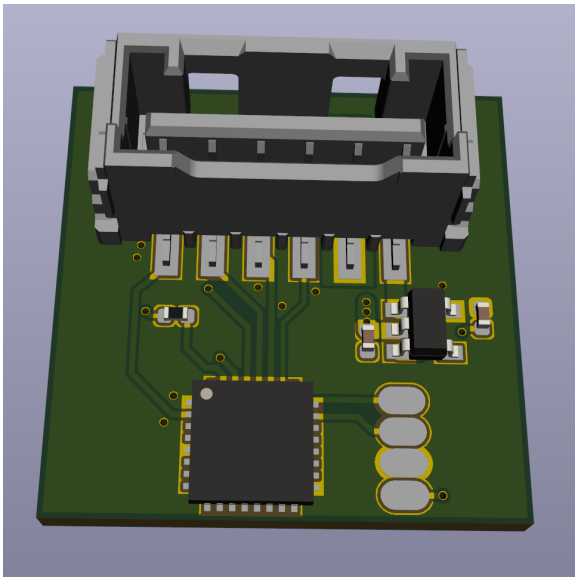


Figure 63: IMU Breakout

The IMU breakout allows a BNO055 IMU to be used as any other device in the system, Using the same communication protocol and similar command IDs as the motors, charger, tail controller and other boards in the system. The board continuously samples the BNO055 IMU for Accelerometer, Gyroscope and Euler angles for heading, roll and pitch and on a request for data responds with all 9 of these data sets from the most recent update.This reduces the data request time for the master controller drastically as this IMU is notorious for clock smearing on i2c and having long delays between samples.

5.2.6 Foot Sensors

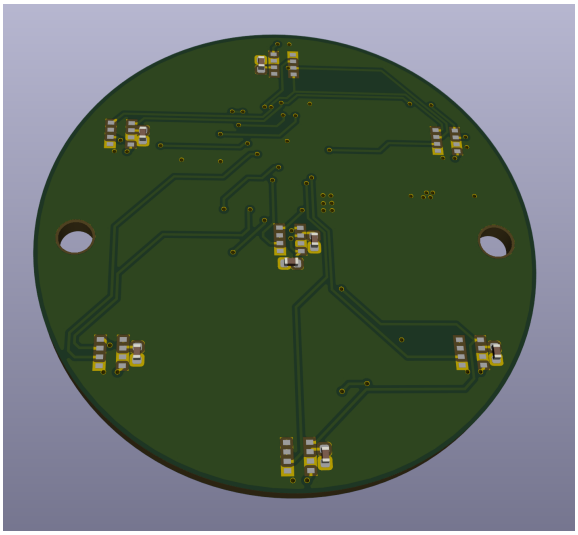


Figure 64: Foot Sensor

5.2.6.1 Board Design

Vector Triangulation Many means of calculating the location of the pressure vector were considered however many of them were unreliable and/or difficult to implement on a micro controller using C/C++. This led us to developing a triangulation algorithm similar to that used by police to triangulate a cell phone based on near by cell towers. The pressure of the sensors in a quadrant(one sector of the total sensor) is read and the equations of 3 circles with pressure dependent radius are calculated, the higher the pressure read the smaller the radius of the circle. The intersection points of the circles are then calculated and the equations of 3 lines are found and the intersection point of all three of those is found. This gives the center point of the pressure centroids.

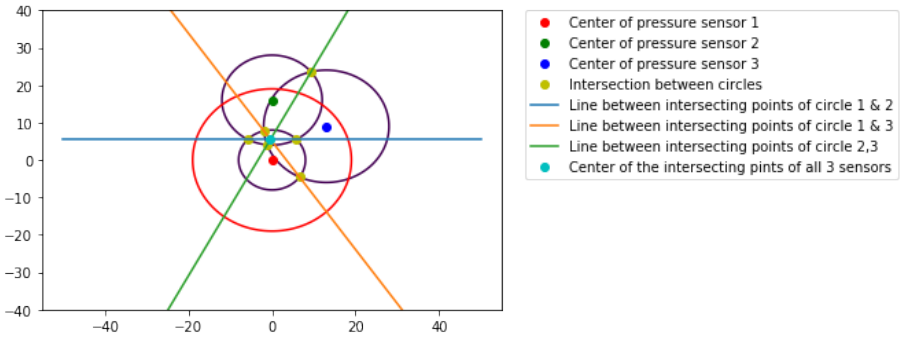


Figure 65: Pressure Vector Triangulation

This location in XY is then extrapolated to the position on the foot by calculating its Z location based on the equation of the sheer created by the casting. The magnitude of the force can also be calculated by the absolute additional pressure introduced to the system by finding out the amount of pressure being applied to all the sensors and removing that from the calculations. essentially finding the minimum pressure that must be applied at each sensor to intersect at that point.

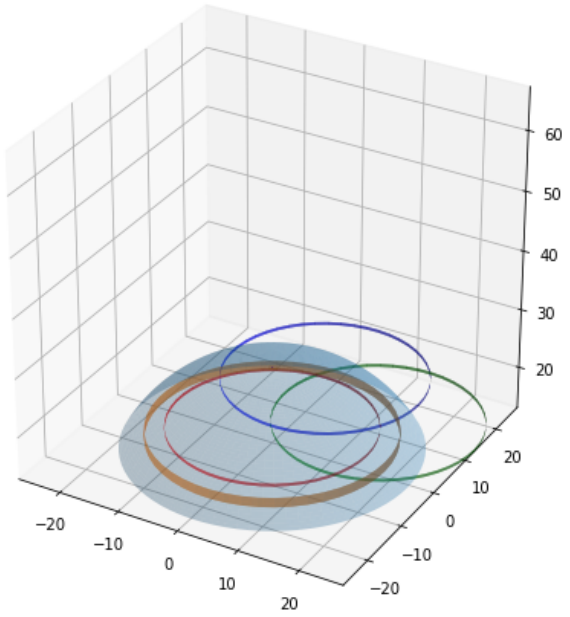


Figure 66: 3D Pressure Vector Triangulation

5.2.7 Charger

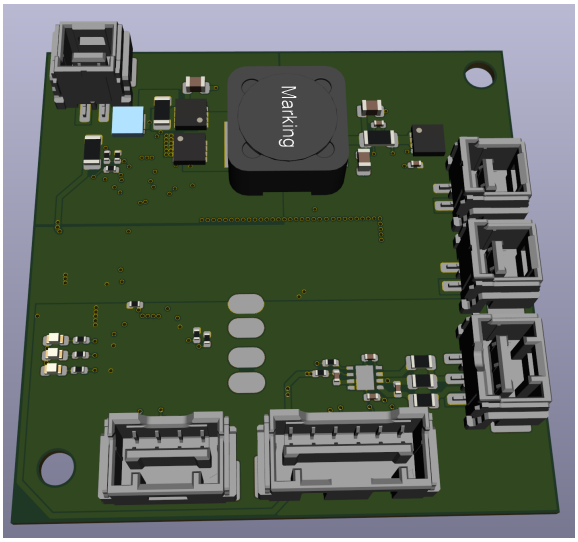


Figure 67: Battery Charger

The development of this charging board allows us to fine tune the charge voltage and charge current of the battery back. This board also allows us to monitor the battery voltage and prevent over discharging as well as balancing the cells at all times. The voltage is constantly being monitored and reported to the motherboard. This allows the system to stop allowing motor updates at a given voltage cutoff point. This safety feature will help to prevent over draw from the batteries diminishing the available capacity of the pack. The charging current is able to be fine tuned up to 8A using BQ24715 charging IC from TI. The charging voltage and current can be updated on the fly through the SMbus interface. The BQ29200 balance charging IC allows for the batteries to be balanced continually ensuring the cells never go out of sync, ensuring longevity of the pack.

5.2.8 Tail Control Board

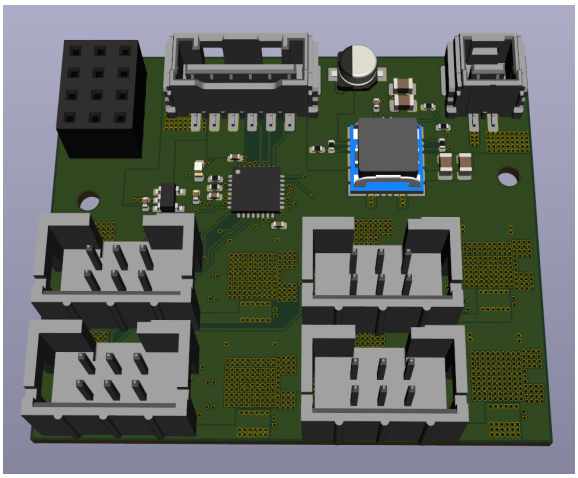


Figure 68: Tail Motor Controller

The intention of this board is to control the 4 Maxon A-Max motors being used to drive the continuum tail of the robot. In order to do this the board must facilitate the homing of the tail, in this case using limit switches, communicate with the motherboard to receive the set point of each motor and run all four PID controllers and keep track of the quadrature encoders for each motor, each having 4096 counts per revolution. This board is also being used to power the on board raspberry pi using an integrated inductor switch mode power supply from TI which is capable of providing up to 60W of power for the case that another SBC was chosen.

5.2.9 SPI to RS485 Converter Board

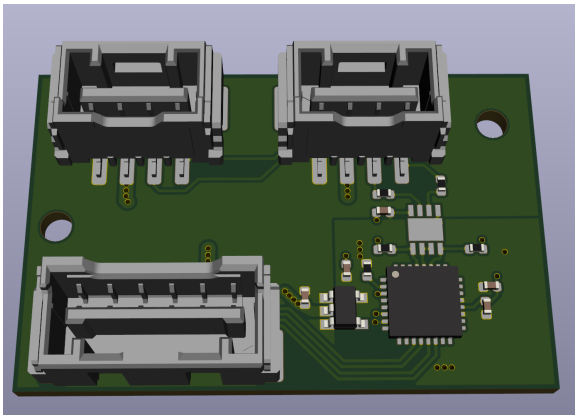


Figure 69: SPI to RS485 Converter

After investing over 100 hours of debugging and testing into the daisy chained SPI protocol of the motors, it was found that The SPI lines were not always pulled completely high or completely low due to the number of devices sharing the same line, this combined with the noise that would be developed in the system and the propagation time between the data being passed between devices made the system very unreliable with 4 motors connected in a chain. A single motor would work 100% of the time, 2 motors would work approximately 80% of the time and decaying exponentially there after resulting in the fourth motor being extremely unstable and completely unreliable. This led to the decision to change from SPI to RS485. Due to time and cost constraints it was decided to leave the motors in the state they currently existed in and add external communication boards, 1 per motor. These boards would take the SPI signal from the motherboard, convert it to an RS485 differential signal that would be sent to all the daughter boards that if the message was meant for that board, would then transfer the data to the motors and the motor response back upstream to the master. This conversion proved to be much more stable dropping nearly 0 packets. This however came with a major drawback, the maximum UART speed of the chosen microcontrollers (LPC824) was only 1Mbit/sec, this combined with the removal of the synchronous protocol meant the re was an effective data transmission speed of only 500kbit/sec. Luckily through testing this proved to be a non issue due to the PID controllers being run locally and the update speed of the USB communications being limited by the computer to only 1kHz

5.2.10 Custom Battery Pack

Figure 70: Custom Spot welded battery pack

After a lot of research the route of creating a custom battery pack made of Panasonic NCR18650B cells was chosen instead of purchasing a commercially available lithium polymer / lithium ion battery. This decision was made for a number of reasons;primarily due to the size and shape constraints of the robot. In order to get a battery pack that would fit in the space we would have to compromise on capacity. Another reason is in doing research Panasonic NCR18650B cells had one of the highest energy to weight ratios available on the market at 3400mAh at only 44g/cell. This allowed us to create a battery pack with a cumulative 13600mAh of power at a nominal 7.4V. This will allow to robot to operate in a walking state for approximately 1.5 hours.

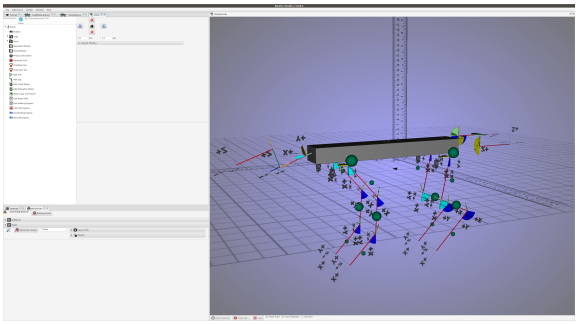


Figure 71: The Implemented Visualization Tool

5.3 Software

As stated in Section 4.3, the software platform chosen was Bowler Kernel, due to its familiarity and visualization capability. During development, we implemented and used a visualization tool to simplify graphics computation on the development machines and to visualize joint limits and DH parameters on the robot. When it came to running a headless system, we also switched from an Odroid XU4 to a Raspberry Pi 3 B+, due to limitations by Bowler Kernel.

5.3.1 Visualization Simplification

When developing software, our team quickly figured out that rendering our complex CAD model for visualization would unnecessarily put computational strain on our computers. Therefore, we implemented a visualization tool that would allow us to look at the constraints of the robot. This included DH parameters, joint limits, and

We also added a flag that would allow us to switch between this tool and the full rendering of the robot for demonstration purposes.

5.3.2 High Level Controller: Odroid vs Raspberry Pi 3

Originally, we wanted to use an Odroid XU4, due to its floating point computation performance. However, when installing and running Bowler Kernel on the Odroid, we realized we needed the affines from the JavaFX library. There was no implementation of JavaFX on the Odroid's processor, and the attempt to install from source did not work. Therefore, we switched to the Raspberry Pi 3 B+, since JavaFX was compatible with it. Although performance was a concern, we were still able to stay under 1ms per cycle.

5.3.3 Our IHMC Implementation

IHMC's library set is a very powerful tool for walking robotics. It includes libraries for dynamic modeling, walking gaits, and simulation for development. The library set has successfully been used on robots including Boston Dynamics' ATLAS and NASA's Valkyrie Humanoid, and was very promising. However, documentation and support quickly became an issue, since the libraries were just beginning to be released and documentation had yet to be published. We did attempt to

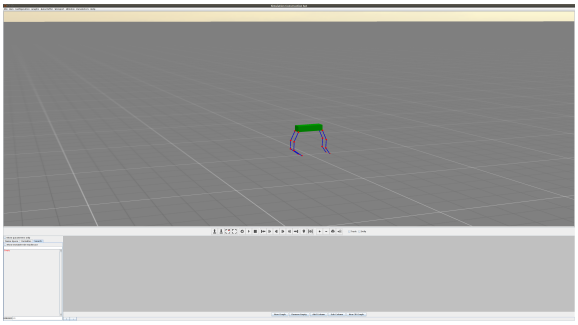


Figure 72: Our IHMC Implementation

develop our software with this system, but it became clear that moving forward would become increasingly more difficult with less and less documentation. However, Figure 72 shows our simulation environment we created to test our code.

6 Results

6.1 Mechanical Results

After my design iterations, tests and hours of CAD and 3d printing the final result for the mechanical design of the SmallKat is something we were definitely proud of.

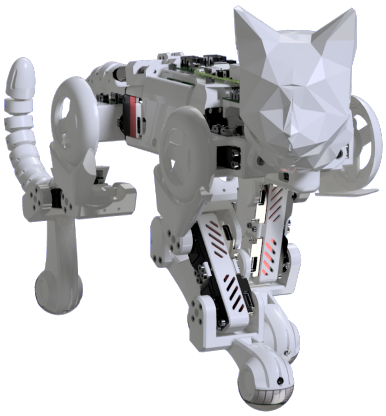


Figure 73: CAD rendering of a front view of the cat

4 DOF legs were created using custom servo motors. These were 3d printed using readily available hardware to allow for simple construction, deconstruction, modification and adaptability. Each leg had an extensive range of motion to allow for both static and dynamic walking gaits. Aesthetic pieces were added to improve the overall look of the robot as well as provide guards for electrical wires. Each foot was equipped with a molded poly-urethane foot that housed the custom

foot sensors. Two different sized legs were made for the front and back of the robot to more closely mimic the actual dimensions and structure of a cat.

A continuum tail was developed to allow for balance control of the cat and add a more realistic look to the robot. After a series of tests with dual extruded designs, the tail was ultimately made using 18 universal joints. This design was split into two individually controlled segments that are each controlled by two Maxon motors and a spring. Each motor has an encoder and is paired with a limit switch that allows the custom tail board to home and control the position of the tail. The kinematics for the tail were tested and implemented to allow for simple control that involved sending two angles to each segment of the tail.

The rest of the mechanical design involved the creation of a body and head. The final design housed all the custom electronics boards, on board computer, battery and sensors. Everything the robot needs to run is on board allowing for future work to be done completely untethered. The head of the robot was designed with two degrees of freedom allowing it to work with the tail to provide some stability and balance control. The same model was used for this design as was used in previous versions of SmallKat. The only difference being size. The head of this design is big enough to house additional cameras and sensors for future iterations of the project.

A full mass analysis of the robot was done to determine center of masses for each individual link and the system as a whole. By applying densities of our specific materials in CAD we were able to approximate the mass to be around 9.6 pounds. The actual mass of the robot weighed in at around 9.4 pounds before all the wires were put on. After wiring the robot weighed in at about 9.7. The discrepancy is most likely due to some error in the densities of our PCB boards and 3d printed parts. However the masses were close enough to give us confidence in the position of the center of masses for each component on the robot. This data was put into Bowler Studios for future work into dynamic walking gaits. The position of the center of mass of the whole robot was also used to place the IMU.

6.2 Electrical Results

Custom Servo motor controllers were developed to fit within the existing footprint of the JX-HV5932MG servos that allowed for position, Velocity and torque control along with live tunable PID, coriolis and gravity gains to allow for smooth control and operation.

A series of pressure sensing foot sensors were developed to allow for triangulation of the force vector being applied to each foot of the robot. This will allow during future development to determine when and where contact has been made during a step cycle in a gait.

A 9DOF IMU board was developed to use the communication protocol of the rest of the robot. By utilizing the BNO055 IMU, a number of data sets are able to be collected, the standard sets of gyroscope, magnetometer and accelerometer in X, Y & Z along with the gravity vector with respect to the IC and the Euler angles roll, pitch & heading. This data will allow for the dynamics of the system to be obtained quickly and accurately to be used in computing the corrective gait cycles.

An integrated charging board that reports the status of the battery pack to the motherboard and in turn motors (for accurate force calculation) was developed in order to easily charge the battery pack with in the robot at the highest charging current permitted by the specifications of the battery pack and reporting back to the motherboard in order to prevent from over draw of the system, which would result in damage caused to the battery pack.

The motherboard developed for this robot handles all communication through out the system including communication to and from the motors, IMU, foot sensors and battery charger. The

motherboard also handles communication back and forth to the integrated on board computer through USB HID where all the kinematics and gaits are processed using data from both the updated trajectory pose and the information returned from the system.

6.3 Software Results

A successful implementation of a static walking gait was achieved using Bowler Studio, Despite this a great deal of time and effort was put into developing both a static and dynamic walking gait in IHMC using integrated libraries and tools in increase the optimization at time of computation as well as to generate the most optimal step cycle, however this was abandoned before completion due to a lack of time.

6.4 Overarching Results

In the end an aesthetically pleasing 4 DOF quadrupedal platform was successfully designed using a series of custom sensors, motors and accessory boards. This combined with the successful implementation of a static walking gait and improved kinematics to utilize the 4DoF of each leg for an orientation angle allowed us to begin the tuning of the walking gait on the robot. While to robot did not quite walk unsupported during the duration of the this project, it is setup to do so. In combinations to the static walking gait developed, a more dynamic gait began development in both Matlab and IHMC using their available tools and libraries for optimization.

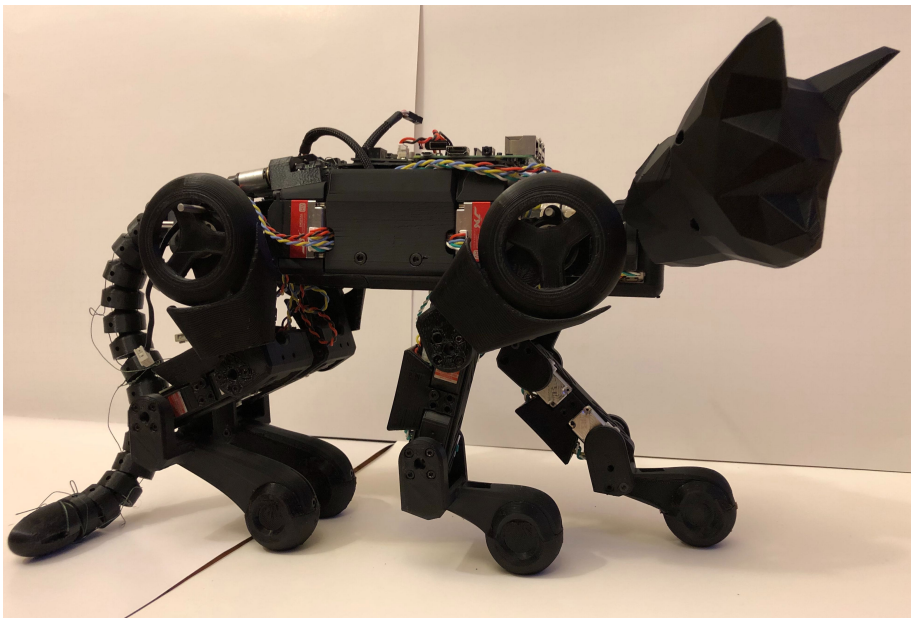


Figure 74: Final Robot

7 Conclusion and Future Work

The project was overall a success despite the number of problems that arose through out the development of the system. A series of custom motors, sensors and accessory boards were successfully developed along with a custom communication protocol. Higher level kinematics and gait control was achieved using Bowler Studios for the implementation of the static walking gait along with the beginnings of a dynamic gait in Matlab and IHMC. The mechanical system resulted in a successful implementation of 4 DOF legs as well as a continuum tail used for dynamic balance. In end there is a great deal of work to be continued on the platform, including successfully integrating all the sensors, making a more robust communication structure and finalizing the tuning of the gaits and the overall system before testing and implementing the dynamics of the system fully. For future research and testing that should be done, there was a lack of information published about the advantages and disadvantages of using a 4DOF leg over a 3DOF leg in a quadruped.

8 Glossary

DMA	Direct Memory Access is a feature of computer systems that allows certain hardware subsystems to access main system memory
DOF	Defrees of Freedom
SBC	Single Board Computer
SPI	Serial Peripheral Interface
IMU	Inertial Measurement Unit
CPR	Counts Per Revolution
CPG	Central Pattern Generator
WSM	Wide Stable Margin
PLA	PolyLactic Acid
IC	Integrated Circuit
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
ROS	Robot Operating System

References

[1] Andreas Aristidou and Joan Lasenby. "FABRIK: A fast, iterative solver for the Inverse Kinematics problem". In: *Graphical Models* 73.5 (2011), pp. 243–260. ISSN: 1524-0703. DOI: https://doi.org/10.1016/j.gmod.2011.05.003. URL: http://www.sciencedirect.com/science/article/pii/S1524070311000178.

[2] Gerardo Bledt et al. "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot". In: Oct. 2018. DOI: 10.1109/IROS.2018.8593885.

[3] Meng Yee Chuah, Matthew Estrada, and Sangbae Kim. "Composite force sensing foot utilizing volumetric displacement of a hyperelastic polymer". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 1963–1969.

[4] Meng Yee Chuah and Sangbae Kim. "Enabling force sensing during ground locomotion: A bio-inspired, multi-axis, composite force sensor using discrete pressure mapping". In: *IEEE Sensors Journal* 14.5 (2014), pp. 1693–1703.

[5] Yasuhiro Fukuoka, Hiroshi Kimura, and Avis H. Cohen. "Adaptive Dynamic Walking of a Quadruped Robot on Irregular Terrain Based on Biological Concepts". In: *The International Journal of Robotics Research* 22.3-4 (2003), pp. 187–202. DOI: 10.1177/0278364903022003004. URL: https://doi.org/10.1177/0278364903022003004.

[6] *Gait*. Sept. 2018. URL: https://en.wikipedia.org/wiki/Gait.

[7] Carlos García-Saura. "Central Pattern Generators for the control of robotic systems". In: *CoRR* abs/1509.02417 (2015). arXiv: 1509.02417. URL: http://arxiv.org/abs/1509.02417.

[8] *Inverse Kinematics*. Mar. 2019. URL: https://en.wikipedia.org/wiki/Inverse_kinematics.

[9] *Jacobian matrix and determinant*. Apr. 2019. URL: https://en.wikipedia.org/wiki/Jacobian_matrix_and_determinant.

[10] *Moore–Penrose inverse*. Mar. 2019. URL: https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse.

[11] *Nybble: World's Cutest Open-Source Roboitc Kitten*. URL: https://www.indiegogo.com/projects/nybble-world-s-cutest-open-source-robotic-kitten/.

[12] Anders Lau Olsen and Henrik Gordon Petersen. "Inverse kinematics by numerical and analytical cyclic coordinate descent". In: *Robotica* 29.4 (2011), pp. 619–626. DOI: 10.1017/S026357471000038X.

[13] Monolithic Power. *12-Bit, Digital, Contactless MA702 Magnetic Encoder Datasheet*. https://www.monolithicpower.com/pub/media/document/m/a/ma702_r1.0.pdf.

[14] *QRP: Quadruped Robotics Platform*. URL: https://web.wpi.edu/Pubs/E-project/Available/E-project-042518-003336/unrestricted/QRP_FinalReport.pdf.

[15] *RoboDog: A Low-Cost Electrically Actuated Quadruped*. URL: https://web.wpi.edu/Pubs/E-project/Available/E-project-042717-152039/unrestricted/MQP_RoboDog_Final_Report.pdf.

[16] Nikolai F. Rulkov. "Modeling of spiking-bursting neural behavior using two-dimensional map". In: *Phys. Rev. E* 65 (4 Apr. 2002), p. 041922. DOI: 10.1103/PhysRevE.65.041922. URL: https://link.aps.org/doi/10.1103/PhysRevE.65.041922.

[17] Bosch Sensortec. *BMP280 Datasheet*. https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-19.pdf.

[18] Bosch Sensortec. *BNO055 Datasheet*. https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST_BNO055_DS000_14.pdf.

[19] *SpotMini*. URL: https://www.bostondynamics.com/spot-mini.

[20] STMicroelectronics. *STM32H43IIT Datasheet*. https://www.st.com/resource/en/datasheet/stm32h743ii.pdf.

[21] STMicroelectronics. *STM32L432KB Datasheet*. https://www.monolithicpower.com/pub/media/document/m/a/ma702_r1.0.pdf.

[22] Yaroslav Tenzer, Leif P Jentoft, and Robert D Howe. "Inexpensive and easily customized tactile array sensors using MEMS barometers chips". In: *IEEE Robot. Autom. Mag* 21.3 (2014), pp. 89–95.

[23] Zhipeng Wang et al. *Your gateway to world-class research journals*. URL: https://journals.sagepub.com/doi/full/10.5772/54051.

[24] Sina Youssefian, Nima Rahbar, and Eduardo Torres-Jara. "Contact behavior of soft spherical tactile sensors". In: *IEEE sensors Journal* 14.5 (2014), pp. 1435–1442.

A Github Repository

https://github.com/SmallKatMQP