**Abstract**

**Acknowledgements**

This project would not have been possible without the help and guidance of the following people:

**Executive Summary**

SmallKat will be designed to fill a void in the research and development of multipedal robotic systems. Currently, quadruped dynamics and gaits are developed using simulation of robotic platforms costing upwards of $100,000. This becomes very impractical for small companies, universities, and hobbyists alike due to the high fixed cost. SmallKat is intended to provide a quadrupedal platform to help research and design new gaits, test sensors and motors, and teach up-and-coming engineering students. 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. At the same time, SmallKat is attainable by private hobbyists who are interested in learning more and developing more complex gaits for quadrupeds. This will exponentially increase the research on quadrupedal walking systems industry and academia alike. SmallKat will be designed to be an adaptable quadruped. Therefore, the system must be highly modular and easily repairable. To allow for a multitude of different gaits, SmallKat will have four 4 degrees-of-freedom (DoF) legs instead of the normal 3 on other platforms. Each joint will be controlled by a powerful servo motor, with possible spring assistance to reduce the current draw of the motor. These motors will be a modified version of standard hobby servos and will be combined with custom sensors to allow for various forms of feedback including position, velocity, torque feedback from each motor, Triangulated pressure vectors for contact point detection on each foot and body linear velocity, angular acceleration and gyro data. To control the SmallKat we intend on creating and easily usable software platform that allows for a smooth development and testing of the SmallKat system.

## Contents

# List of Figures

# List of Tables

**Authorship**

# 1 Objectives

The purpose of this project is two fold.

### Mechanical Objectives

1.

2. Objective 2

### Electrical Objectives

### Software Objectives

#### 1.0.0.1 Time frame

# 2 Background

## 2.1 Introduction

this is a test

## 2.2 Prior Work

The SmallKat project started in October 2017 as an ISP with professor Ciaraldi. The original intention of the project was to create a walking quadruped in a single term. The results of this project while being quite successful, left us wanting a much better solution, there were problems with the manufacturing of the servos and a lack of any feedback from the system. During the summer of 2018 we continued to develop and improve the platform. SmallKat V2 used much more reliable servo motors and several sensors including IMUs, pressure sensitive feet and current sensing. However due to time constraints these sensors have yet been integrated into the control system of the quadruped. Instead a reliable static walking gait was created and tested. But the lack of feedback from the servos prevented any further work on more complex walking gaits. 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. This includes using 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.

## 2.3 Boston Dynamics

During the course of the project we reached out to Boston dynamics and arranges 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 $\tilde{1}$ 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 gittering and jerkiness from the system.

# 3 Research

## 3.1 History of Quadrupedal Robotics

## 3.2 Basic Dynamics in Quadrupeds

## 3.3 Gaits

### 3.3.1 What is a gait

A gait is the motion of taking a step. This includes the motion of each joint, the time at each place, the speed of motion and the time between cycles. In the case of a quadruped robot, the

### 3.3.2 Types of gaits

## 3.4 Notable Quadrupedal Robots

### 3.4.1 Boston Dynamics' Spot Mini

### 3.4.2 MITs Cheetah

## 3.5 SmallKat V1 & V2

### 3.5.1 Electrical

### 3.5.2 Computer Science

### 3.5.3 Mechanical

# 4 Methodology

## 4.1 Electronics Design

### 4.1.1 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

### 4.1.2 Motors

Many motor solutions were considered when deciding for this project. These include the common hobby servo, specifically the Jx-Servo HV-5932MG due to its common voltage rating and its availability in a high torque metal geared variant, The Robotis Dynamixel Smart Servos, both the AX and XH series were considered. The Pros and Cons of each motor is listed below.

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

Table 1: Pros & Cons of each motor

Each of these motors was researched and then the solution 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[**MA702**].

#### 4.1.2.1 Frequency

When designing the motor controllers 2 hardware timers were used for Pwm pins which would allow for up to $\tilde{4}$MHz pwm signals to be used. In the preliminary stages of testing an arbitrary value was chosen which equated to $\tilde{1}$MHz 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 $\tilde{5}0\%$ 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.

#### 4.1.2.2 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.

#### 4.1.3 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[**STM32H43IIT**] and the STM32L432kb[**STM32l432KB**]. 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 ann 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.

#### 4.1.4 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 STMicroelectronics, Bosch Sensortec and many other were considered but primarily the LSM303CTR, LSM6DS3USTR, BMX055, BMI160 and the BNO055[**BNO055**] 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.

#### 4.1.5 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[**chuah2012composite**][**chuah2014enabling**], WPI[**youssefian2014contact**] and MIT[**tenzer2014inexpensive**]. The research done by the group for the paper "Inexpensive and Easily Customized Tactile Array Sensors using MEMS Barometers Chips" [**chuah2012composite**] 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 [**BMP280**] 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.

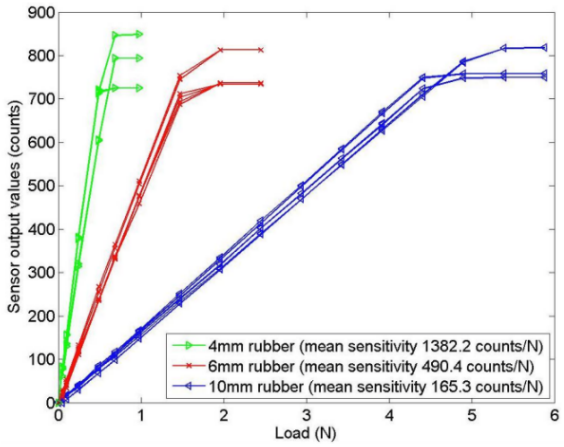


Figure 1: Correlation between load and pressure readings at different thicknesses of polyurethane[**chuah2012composite**]

The research done by the Contact Behavior of Soft Spherical Tactile Sensors[**youssefian2014contact**] 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.

#### 4.1.6 Communication Protocol

##### 4.1.6.1 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<br>Easy Daisy chaining<br>Available on most micro controllers<br>Supports DMA | Requires independent addresses on each device<br>Requires independent firmware to be flashed to each motor<br>Non synchronous<br>Requires motor to be re flashed to move to a different location |
| SPI | Synchronous<br>Max baud rate of 16.5 Mbaud<br>Supports DMA<br>Available on most microcontrollers<br>Daisychainable<br>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<br>Simple 2 Wire interface | Not available on most micro controllers<br>Slow |
| RS485 Serial | Simple to use<br>Designed to daisy chain | Requires external IC<br>Slow<br>Requires a software address be set |
| RS232 Serial | Available on most micro controllers<br>Simple to use<br>No hardware addresses needed | Slow<br>non synchronous<br>Difficult to daisy chain but possible |

Table 2: 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.

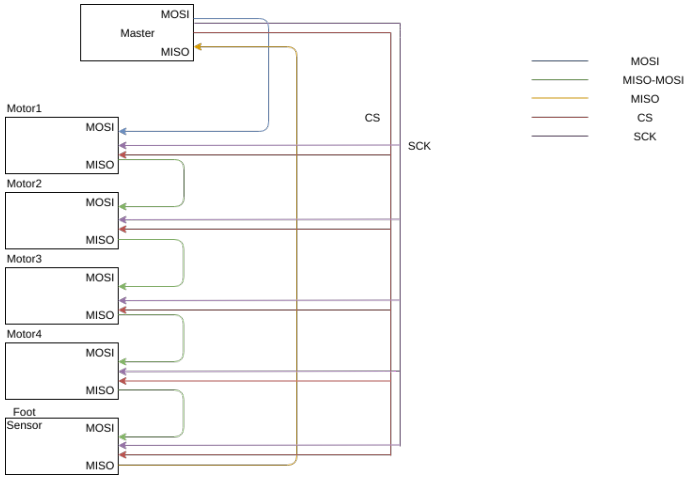


Figure 2: Flow of data through a leg

#### 4.1.6.2 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 $\tilde{2}$ 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 Software and Controls

### 4.2.1 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.

### 4.2.2 Master Controller

This controller 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 3: 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 4: 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<br>• Motor 2 Receives return data from Motor 1 and replies its data to Motor 3<br>• Motor 3 Receives return data from Motor 2 and replies its data to Motor 4<br>• Motor 4 Receives return data from Motor 3 and replies its data to the foot sensor<br>• 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<br>• Motor 2 Receives the Foot Sensor packet and replies Motor 1 data to Motor 3<br>• Motor 3 Receives return data from Motor 1 and replies Motor 2 data to Motor 4<br>• Motor 4 Receives return data from Motor 2 and replies Motor 3 data to the foot sensor<br>• 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<br>• Motor 2 Receives the Motor 4 packet and replies the Foot Sensor packet to Motor 3<br>• Motor 3 Receives the Foot Sensor packet and replies Motor 1 data to Motor 4<br>• Motor 4 Receives return data from Motor 1 and replies Motor 2 data to the foot sensor<br>• 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<br>• Motor 2 Receives the Motor 3 packet and replies the Motor 4 packet to Motor 3<br>• Motor 3 Receives the Motor 4 packet and replies Motor 1 data to Motor 4<br>• Motor 4 Receives the Foot Sensor packet and replies Motor 1 data to the foot sensor<br>• 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<br>• Motor 2 Receives the Motor 2 packet and replies the Motor 3 packet to Motor 3<br>• Motor 3 Receives the Motor 3 packet and replies Motor 4 packet to Motor 4<br>• Motor 4 Receives the Foot Sensor packet and replies Motor 1 data to the foot sensor<br>• 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 5: Order of Data returned from a leg

#### 4.2.3 Motors

When the motor receives a packet it checks if the first byte if the packet and compares it to the Dev 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 setpoint in Postion PID loop |
| 0x92 | Updated position setpoint in Velocity PID loop |
| 0x93 | Updated position setpoint in Torque PID loop |

Table 4: Command values for servo

In between receiving packets the motors run multiple PID loops simultaneously to control position, torque and velocity at an 161kHz refresh rate.

#### 4.2.4 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 5: Command values for IMU

#### 4.2.5 Foot Sensor

Smurlarly 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 6: Command values for Foot sensor

### 4.2.6 High-Level Control Software

The High-Level Controllers' jobs' to run the kinematics and dynamics. This will include balancing, foot-step planning, and all the gaits we are planning on developing. The software written must have a few key objectives to hit to be considered useful for this application.

1. Real-time performance with a guaranteed response time of under 2ms
2. Open-source and easily modifiable if needed
3. Able to communicate quickly to our custom components and off-the-shelf components, like cameras, IMUs and other microcontrollers.

With this in mind, there were a few options we found.

### 4.2.7 Software Platforms

#### Bowler Studio

Bowler Studio is a robot development application that combines scripting and device management with powerful control and processing features. It is comprised of two major parts: Bowler Studio, a development and visualization tool, and Bowler Kernel, a low-level platform designed to run at high cycle rates on Linux machines. The entire system is developed in Java, giving us developmental flexibility when it comes to operating systems, Linux distributions, or hardware platforms. It also improves development speed, since a library package manager like Gradle or Maven can be used. Previous versions of SmallKat were developed with Bowler due to its quick bring-up period. However, there are a few major shortcomings for Bowler Studio. The first is its adoption. Both Bowler Studio and Bowler Kernel are not widely used in the field of robotics. Using a platform like this will push its stability and performance to the max.

#### 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 functions, 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 2 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, over 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 sockets instead of 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 2 years old in 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 best). 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.

**Custom Platform**

Since neither ROS 1/2 nor Bowler Studio is exactly what we were looking for, we considered writing a custom platform. The platform would be written in C and C++ and would likely communicate between processes using named pipes. This would allow for cross-platformability between Linux, Windows, and macOS, because every modern operating system has some implementation of named pipes. Communicating through named pipes is also extremely fast, because it is just reading and writing to files. The other communication method discussed was sockets. The main benefit is that they can be easily integrated for communication over Ethernet or WIFI. However, developing a custom platform is complicated and adds a lot of work and development time to ensure stability and functionality.

#### 4.2.8 High-Level Controller

In the end, we decided to use Bowler Studio for the High-Level Controller. It offers cross-platformability and easy development due to its Java backbone. It also is relatively stable although not used much in the industry.

#### 4.2.9 Localization and User Control Software

For the Localization and User Control Software, we decided on using ROS 1 due to its stability and wide range of open-source packages. The Localization and User Control Software will be split into two major sections: Visualization and Mapping, and User Control.

**Visualization and Mapping**

Visualization and Mapping will be utilizing the multitude of SLAM and navigation algorithms built into ROS 1. It takes in input from the on-board camera, and provides direction and velocity to the High-Level Controller. This allows us to easily implement a fully autonomous robot capable of exploring new areas.

**User Control**

The user control feature will have two parts: control and supervision. The control part will allow users to initiate gaits and control them remotely from another computer. It will also allow for initiating the autonomy function built in to SmallKat. The supervision portion will give the user the capability of watching different key variables, like joint angles, relative position, and body position relative to the world. This will help with debugging while developing on SmallKat.

### 4.3 Mechanical Design

#### 4.3.1 Design Considerations

- 

#### 4.3.2 Controls Equations

#### 4.3.3 4 dof

#### 4.3.4 Motor Tests

## 5 Development and Implementation

## 6 Results

## 7 Conclusion and Future Work

## 8 Conclusion

## 9 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 |