Low Cost Quadruped: MUTT

A Major Qualifying Project Proposal submitted to the Faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degrees of Bachelor of Science in:

Robotics Engineering,
Mechanical Engineering,
Computer Science.

Joshua Graff
Alonso Martinez
Kevin Maynard
Alexandra Bittle

Date:4/26/2017
Professor William Michalson, Advisor

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects.

Abstract

The field of educational and research robotics is alight with development platforms that fall short of being interesting and novel. Our goal was to create a quadruped for use as an entry level research project for students and educators. Reducing cost through the use of commercially available parts combined with rapid-prototyping, we built a platform that can be used to teach and learn legged locomotion for less than $600 (half the price of a Turtlebot 2 from OSRF). Our robot was able to walk in basic form using limited actuation; this was limited by the components we chose - specifically the motor controllers for part of the actuation. We expect that using components better suited to the task could accomplish what we set out to achieve.

Executive Summary

Introduction

The field of educational robotics both in higher education and personal research focuses heavily on the following two areas: wheeled, holonomic robots and multi-DOF (Degree of Freedom) serial manipulators. Experimentation in fields outside of these areas requires a significant investment of experience and resources. By decreasing the required investment, we reduce the cost of entry to legged robotics. Combining low cost components with a modular design, we designed a quadruped whose primary purpose is to be an advanced research tool.

Novelty

The current domain of research robotics for undergraduate students is dominated by two types of robots: hardware produced for commercial sale, and low quality handmade hardware. Legged robots haven't yet found their way into undergraduate labs. The absence of these robots is a key tell on the progress of coursework in the area. For two reasons, legged robots have been unavailable in the undergraduate domain, the lack of good hardware, and the challenge required to program such devices.

In developing a low-cost hardware solution, we hoped to solve both of these issues at the same time. By integrating commercial off the shelf parts and an easy to produce rapid prototyping components, our robot is able to be made and modified by anyone with 3D printing experience. Furthermore, using controls systems that require entry level skills makes programming the robot accessible to a large audience. Producing the device with components familiar to anyone with basic experience was key to making it approachable.

Design

The hardware design is focused around building a device that is inexpensive to produce and maintain, and powerful enough that it has room to grow. By prioritizing parts that can be manufactured with rapid-prototyping machines and commercial off the shelf components, we proposed a system that could be built and programmed by anyone with minimal application-specific knowledge.

We began our design work with an initial Solidworks model and evaluated its feasibility using kinematic and dynamic analyses. These analyses allowed us to predict the motion of the robot's legs and the resultant torque the actuators must overcome. Once our design was verified, we began manufacturing components. By obtaining commercial off-the-shelf parts for fastening, mounting, control, and computation, we were able to increase production rates at minimal cost.

Integrating “Sense, Think, Act” into our robot is a key goal to making a device useful to robotics research. Developing hardware capable of joining the three, and a software suite to demonstrate is the basis of the design.

Results

Although initial testing with using Pololu motor controllers showed that they would be sufficient, they ultimately entered failure state due to overcurrent and were inadequate to drive the VEX 393 motors. As a replacement, Vex 29 motor controllers were chosen for their availability and after passing initial tests for current handling. In practice, however, the Vex 29s did not stand up to the demands of the Mutt robot. In the end, the upper limbs became inactive to focus on the rest of the system.

The lower joint of each limb was actuated by a series elastic tendon controlled by a high-torque servo motor mounted on top of the robot. Upon initial testing, the servos reached their maximum torque while driving the lower limb. Increasing the force on the tendon by reducing the radius of the cams resolved this issue.

Although the project initially planned on using an Arduino Due, limited ability for I2C communications with daisy-chained encoders caused us to use an Arduino Mega. This issue could have been caused by many reasons, but we believe that the Arduino Due was unable to provide the necessary voltage and current to communicate with four encoders. Overall, the performance of the Arduino Mega was suitable for the robot’s needs. The team was surprised and impressed with the performance of the Arduino Mega. Operating with a 10ms interrupt rate, the device was more than capable of handling the tasks required for smooth operation of the robot.

The circuits designed and built functioned as expected, after a short debugging period. The hardware emphasized modular design and minimized the number of connection points; this made the device easy to maintenance and robust during testing. Another

important feature of the circuitry was protection installed on the power bus. The primary purpose of this was to protect the logic circuitry from the electrical noise generated by the motors and servos. After adding the capacitance and changing control boards to a more robust Arduino Mega 2560 zero components were damaged or destroyed.

The gaits we produced to demonstrate the ability of the robot were limited by the quality of the components we chose for our actuators. The robot was still able to demonstrate its potential with partial functionality operational. Demonstrating the ability of the robot to move with only partial actuation proves that the system has the potential to operate at full capacity with moderate changes to the system.

In order to demonstrate the functionality of the robot, we created an Android app to communicate between an Android phone and the robot. The app had a series of buttons that, when pressed, transmitted a Bluetooth message to the onboard Raspberry Pi. The Raspberry Pi then transmitted the message to the Arduino through serial communications. This process did not have a noticeable amount of delay in transmitting messages and was simple and effective for showing off the robot's functionality.

Future Work

Future work on this project should revolve around replacing the motor controllers and motors. As previously stated, the robot was incapable of moving the upper limbs. We hypothesize that the root cause is a combination of factors causing the motors to be unable to produce the force required. The motors chosen were specified to produce what was thought to be enough force to actuate the robot, in reality, those forces were likely much higher than expected. In order to cut costs, the motor controllers that were used were cheap and were being run at voltages higher than they were rated. Because of this, the controllers often entered failure modes unable to drive the motors. Replacing the actuators and drivers on the upper limb is key to making the robot fully functional.

Furthermore, additional work can be done on the Matlab scripts that generate the trajectories. Although the scripts were useful in visualizing the trajectories, joint angle transformations between the Matlab visualization and the robot movements was not entirely accurate. Refining the motion planner, developing additional planning and control tools, and better integrating them with the robot are important improvements to make the robot more effective.

Conclusions

Solutions for legged robotics research fall into one of two categories: too simple to be of merit, or too expensive to be available to novice roboticists. Our project sought to eliminate both of these barriers with an inexpensive solution to legged robot research. We designed and constructed a robot whose programming interface is simple and available to all, and whose hardware was readily reconstructed from inexpensive components. Through this challenge, we demonstrated the effectiveness not only of modern low-cost computation hardware, but also the ability to develop an accessible solution to a problem typically reserved for well-funded research laboratories.

Table of Contents

Abstract 2
Executive Summary 3
Introduction 3
Novelty 3
Design 3
Results 4
Future Work 5
Conclusions 6
Table of Contents 7
Table of Figures 12
1. Introduction 14
2. Background 16
2.1. Why Legs? 16
2.2. Current Field of Quadrupeds 17
2.3. Project Novelty 19
2.4. Current Solutions and Assumptions 20
2.4.1. Mathematical Modeling 20
2.4.2. Materials 21
2.4.3. Joint Actuation 21
2.4.4. Computation 22
2.5. Current State of the Art 23
3. Project Strategy -- Design Specification 27
3.1. Hardware Design 27
3.1.1. Material 27
3.1.2. Actuation 28
3.1.3. Electrical Design 29

Power Distribution 29
Circuit Design 31
Sensor Package 31
3.2. Control System Design 33
Embedded Controls System 33
Input and Behavior Controls System 34
3.3. System Analysis 34
3.3.1. Kinematics 35
3.3.2. Dynamics 36
4. Specific Application Parameters 38
4.1. Controls Processing 38
4.2. Electrical Subsystem Design 39
4.3. Hardware Architecture Design 40
4.3.1. RaspberryPi 3 41
4.3.2. Arduino Mega 41
4.3.3. Base Station 41
4.4. Software Architecture Design 42
4.4.1. Software Flow 42
4.4.2. Operating System 43
4.4.3. SLAM 43
4.4.4. Image Correction 43
4.5. Gait 44
4.6. Behavioral Control 44
4.7. Bill of Materials 46
5. Application 47
5.1. Rapid Prototyping 47
5.1.1. Appropriate Clearance for Parts 47
5.1.2. Part Design for Rapid Prototyping 47

5.1.2.1. Original Design 47
Body 47
Legs 48
Nacelle 49
5.1.2.2. Final Design 49
Body 50
Legs 51
Nacelle and Cam 51
5.2. Interfacing Rapid Prototyping and COTS 52
Shafts and Bearings 52
Cables and Fasteners 53
5.3. Actuator Testing and Integration 53
5.3.1. Initial Test Results 53
5.3.2. Test Results and Redesign 54
5.4. Controls Programming 55
5.4.1. Application of Control System 55
5.4.2. Trajectory 55
5.4.3. Gait Programming 56
5.5. Electrical Systems 57
5.5.1. Electrical Design 57
5.5.2. Wiring Design 57
5.6. Camera Correction Software 59
Code Implementation 59
5.7. Raspberry Pi 3 Setup 60
5.7.1. Raspbian Jessie 60
5.7.2. Ubuntu MATE 60
5.7.3. ROS Integration 61
5.7.3.1. ROS Integration on Raspbian Jessie 61

5.7.3.2. ROS Installation on Ubuntu MATE 61
5.7.4. Operating System Installation and Configuration 62
5.7.5. Swap Space 63
5.8. GMapping 63
GMapping on MUTT 65
5.9. VisualSLAM 66
5.10. BreezySLAM 66
5.11. HectorSLAM 67
5.12. Voice Commands 69
Speech Parsing 69
Alexa Integration 70
Integrating Alexa with the Raspberry Pi 70
Setting up the Alexa Skill 70
Utterances 71
Communicating with AWS and Alexa 72
Setting up the Communication Server 72
5.13. Navigation 73
Navigation with Alexa 74
Patrol Mode 74
5.14. Facial Recognition & Following 74
5.14.1. Following 75
6. Results 77
6.1. Actuators 77
6.1.1. Motors and Controllers 77
6.1.2. Servos 77
6.2. Microcontrollers 78
6.2.1. Arduino Due 78
6.2.2. Arduino Mega 79

Clock Speed 79
Electrical Characteristics 79
Architecture and Trajectories 80
6.3. Electrical 80
6.4. Motion and Motion Planning 81
6.5. Mapping 82
6.5.1. GMapping 82
6.5.2. HectorSLAM 84
6.5.3. RGB-D SLAM 84
6.5.4. BreezySLAM 85
6.5.5. Navigation 85
6.6. Control Interface 86
6.6.1. Raspberry Pi to Arduino Communications 86
6.6.2. Android App Control 86
6.6.3. Voice Control through Alexa 87
7. Future Work 88
7.1. Motion Planning 88
7.2. Actuation 88
7.3. Mapping 89
7.4. Voice Control 89
7.5. Mechanical Design 90
8. Project Insights 90
8.1. Mechanical Design 90
8.2. Electrical Design 91
8.3. Software Design 92
8.3.1. Arduino Software 92
8.3.2. Raspberry Pi Software 93
9. Conclusion 94

Table of Figures

Figur 1: Design Requirements 4
Figure 2: The Qinetiq Talon Robot 6
Figure 3: A Google self-driving car goes on a test drive in California (Higgins, 2016) 7
Figure 4: The General Electric "Walking Truck" stood eleven feet tall (Raibert, 1986) 8
Figure 5: Sony Aibo (“Aibo..,” 2016) 9
Figure 6: Ghost Robotics: Minitaur 11
Figure 7: Boston Dynamics SpotMini 12
Figure 8: MIT Cheetah 2 13
Figure 9: Ghost Robotics Minitaur 14
Figure 10: An example application of subsumption architecture 15
Figure 11: Leg Actuator Locations 17
Figure 12: Electrical Diagram 18
Figure 13: Control System Architecture 21
Figure 14: Working Model for Robotic Leg Kinematics 23
Figure 15: Electrical Subsystem layout 26
Figure 16: Robot Subsystems 27
Figure 17: Robot State Diagram 30
Figure 18: Original Robot Design 35
Figure 19: Original Leg Assembly 36
Figure 20: Final Robot Design 36
Figure 21: Final Body Assembly Design 37
Figure 22: Final Leg Assembly 38
Figure 23: Nacelle, Servo, and Cam Assembly 39
Figure 24: Leg testing configuration 40
Figure 25: Redesigned Servo and Nacelle 41
Figure 26: Gait planning graphic output. 43
Figure 27: Molex Microfit and the XT60 Connector 44
Figure 28: TE Connectivity Connector Assembly 45
Figure 29: GMapping Results from Outer RBE Lab Room 50
Figure 30: Mapping from HectorSLAM 52
Figure 31: Facial Recognition Output 53
Figure 32: Electrical Diagram 58
Figure 33: Gait planning output, with numbered legs 59

Figure 34: Initial Android App Design 61

1. Introduction

The field of educational robotics both in higher education and personal research focuses heavily on the following two areas: wheeled, holonomic robots and multi-DOF (Degree of Freedom) serial manipulators. This prominence is evidence of a scarcity of interesting, low cost robotic platforms.

Experimentation in fields outside of these areas requires a significant investment of experience and resources. By decreasing the required investment, we can reduce the cost of entry to legged robotics. Feasibly, this could mean an acceleration of development by creating an environment in which working with advanced robotic systems requires neither a large capital investment nor an advanced degree. Frustrated with the current environment, we seek to address the scarcity of such a system.

Indeed, current platforms for educational robotics represent a solved problem; holonomic drive is a well-defined study. While there are nuances in its application, students working with these systems find themselves regurgitating well established solutions. As students ourselves, we are driven to advance the field of studying robotics so that individuals may apply a more diverse solution set.

Figure 1: Design Requirements

Design requirements (illustrated above) for completion of this goal are as follows:

- A legged robot

- Total cost below $600
- Low barrier of entry to further development
- Walk - The robot will be able to walk at a standard pace along flat or slightly inclined terrain. Uneven or heavily inclined terrain are not within the scope of this project.
- Navigate - The robot will be able to perform SLAM and be able to navigate from one point to another based off of human input.
- Avoid obstacles - As part of its navigation, the robot will be able to avoid obstacles and replan its path if needed to avoid any obtrusions.

Combining low cost components with a modular design, we designed a quadruped whose primary purpose is to be an advanced research tool. The functionality provided by this platform will be a framework on which development can occur. Modularizing the system may make future development easier. Using commonly available components, our robot will be easily adopted by new users.

2. Background

2.1. Why Legs?

The current field of robotics is dominated by static, wheeled, and tracked platforms, in each case connecting the platform to the ground in a known association. This coupling lends itself to a stable platform that is simple to control and -in the case of wheels and treads- offers some ability to traverse terrain. The use of tracks and wheels also exploits the experience of decades of research into manned wheeled mobility. The result being a commanding presence of these mobility systems in products currently on the market of mobile robots, such as the Qinetiq Talon below (Castner, 2013).

Figure 2: The Qinetiq Talon Robot

Tracked robots such as the Talon tradeoff speed for stability and terrain handling. Treads give a robot the ability to traverse diverse terrain, commonly at the cost of speed. Using wheels, a robotic system's speed can be improved in some cases. Specifically in cases with even, static terrain, wheels can increase the speed (and in some cases stability) of a robotic platform. Cars are a good example of this increase in

speed given some assumptions about the terrain; able to reach higher speed, at the cost of being unable to conquer more complex terrain features.

Figure 3: A Google self-driving car goes on a test drive in California (Higgins, 2016)

Where automobiles trade terrain for speed, and tracks trade speed for terrain, legs offer to bridge the gap, providing medium speeds across diverse to challenging terrain. Where wheeled and tracked vehicles maintain (for the most part) a consistent contact with the ground, legged systems are able to be mostly independent from the specific roughness of the ground (Raibert, 1986). “Aside from the sheer thrill of creating machines that actually run,” legs hope to provide effective, powered ground transportation across any terrain (Raibert, 1986).

2.2. Current Field of Quadrupeds

Decades of refinement and engineering have built automobiles to the popularity they currently hold, and to some extent, the world in which they operate has grown around them as much vice versa. Legged vehicles, on the other hand, are still in a mostly investigatory phase. With a still-developing commercial market for pedal robots and vehicles, applications have since tended to be focused in three markets: Research, Military, and Toys; each to a lessening extent.

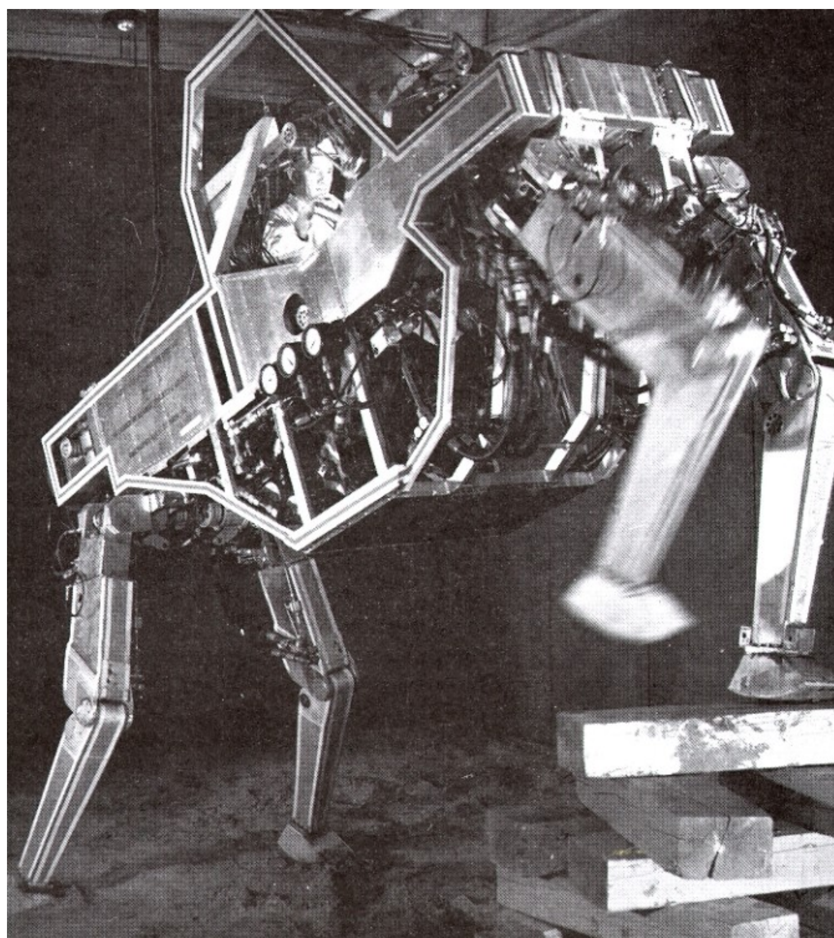


Figure 4: The General Electric "Walking Truck" stood eleven feet tall (Raibert, 1986)

Possibly because of the novelty of the approach, quadrupeds are most commonly found in research laboratories, both public and private. In the 1960’s, Ralph Mosher at General electric built what is known as the “Walking Truck.” The walking truck was a human carrying and controlled machine with impressive, but limited effectiveness. Honda entered in the year 2000 with its Asimo robot, a self-controlling humanoid robot. Early leg actuated machines such as this are known as “static crawlers,” machines whose forward velocity is slow enough to maintain a stable base of support on the ground at all times (Raibert, 1986). Static walking provided a relatively simple starting point to get legged platforms off the ground -- so to speak -- and further developing the field is even until now, done mostly in research laboratories.

Research labs were the first to demonstrate that autonomous mobile platforms could operate using legs. The results produced in research, while effective breakthroughs in a

young field, lacked practical purpose. In 2005, Boston Dynamics (a research firm started by Marc Raibert) produced BigDog: a capable, dynamic walking platform that has been deployed with United States Marines (Diaz, 2014). This BigDog robot, and its Boston Dynamics cousins are examples of research robotics blending into potential military application of pedal robotics. Boston Dynamics regularly accepts funding from government sources such as the Defense Advanced Research Projects Administration (DARPA) and the United States Marine corps to develop potential military applications of robotics.

Figure 5: Sony Aibo ("Aibo..," 2016)

Profitable quadrupeds are yet to find a solid foothold in the commercial market. Attempts at civilian market entry have been made, primarily in the toy and entertainment market. Sony's AIBO pet dog was the main entrant to this field, winning multiple awards for its design and technical featureset. Aibo was sold as a toy, with a complex artificial intelligence driving its decisions and actions ("Aibo..," 2016). Initially sold in 1999, this companion was a demonstration primarily in the interactions it was able to have with people. It demonstrated a potential market that yet remains unexploited after Sony stopped supporting Aibo in 2014.

2.3. Project Novelty

A multitude of platforms exist for relatively low-cost entry into wheeled robotics. However, no such standardized platform has been developed for the experimentation and development of legged robots. While mechanical legs and manipulators may be

found online, there is no standard way that a high level experimentation may be done in a controlled environment.

By creating a modular platform that is primarily focused on 3D printed parts, this robot may drive the cost of experimentation down so that even hobbyist roboticists may learn from its design and manipulation. It will also allow for higher level control and data collection at a much lower cost compared to corporations which spend millions developing such robots.

2.4. Current Solutions and Assumptions

Associated with the advancement of legged robotics are new and more complex problems. These problems include properly modeling systems for accurate design, selecting functional actuators for movement, determining correct sensor packages to recognize the various changes in environmental elements, and creating algorithms to run all of these features, and more. Fortunately, there has been research in all of these areas that help to simplify the design and creation of legged robots.

2.4.1. Mathematical Modeling

To properly guide the development of robotic platforms, a number of mathematical models must be produced. These models include the kinematics, kinetics and dynamics involved with the manipulation of robotic elements (Murray, 1994). The study of kinematics includes the study of motion without regard to forces, whereas kinetics describes the forces acting on a system in motion (Norton, 2012, pp. 3). Both of these areas of study are branches of the larger study of dynamics (Hibbeler, 2016, pp.3). Using the equations for motion derived using these mechanical methods, it is also possible to determine the necessary electrical requirements to power the robotic system. This information can then be used for actuator selection and motion control (Niku, 2011, pp.175).

“One principal aim of kinematics is to create the desired motions of the subject mechanical parts and then mathematically compute the positions, velocities, and accelerations that those motions will create on the parts (Norton, 2012, pp.4).” In robotics, this is completed using two different methods, forward and inverse kinematics. These two computation methods are simple inverses of each other; forward kinematics implements given joint angles to determine end-effector positions, while inverse

kinematics determines the values for the joint angles given desired end-effector position and orientation (Spong, 2004, pp.61).

Finally, following the kinematic analysis of the system, a dynamic torque-force analysis is completed to determine the torques present in moving the mechanism. In this analysis, using given or desired masses, velocities, and accelerations, the reaction torque at each joint is able to be computed (Norton, 2012).

2.4.2. Materials

Proper material selection for the construction of a robot is a major task. A number of significant parameters come into play such as mechanical properties, wear, manufacturability, and cost (Jayakoda, 2011). Metals, plastics, and composites tend to be the most used materials for the structural elements of robots. Metals have very good mechanical properties and certain metals, such as aluminum, stand up well against corrosion. Plastics, while not as strong as metals, can be used very effectively for small robots so long as their temperature remains below roughly 100 C. Plastics are also much easier to machine and their cost can be dramatically less than most metals. Composites, while exhibiting material properties matching or exceeding those of metals, unfortunately are much more expensive, as they are more difficult to manufacture (Department of C.S.: University of Rochester, n.d.).

2.4.3. Joint Actuation

Joint actuation is one of the most important facets of legged robotics. Proper actuators must be selected based on power/mass, torque/mass, and overall efficiency (Hollerbach, 1992). When large robots are being designed, or when a large force output is required, hydraulic actuators excel. Robots like Boston Dynamics BigDog use hydraulics to propel itself over terrain (Boston Dynamics, 2016).

Figure 6: Ghost Robotics: Minitaur

For smaller robots, however, electric motors can prove to be quite powerful. Such an example is Ghost Robotics' Minitaur. This medium-sized robot features powerful outrunner motors that allows it to not only run, but jump over difficult terrain. Each leg is driven by two of these motors that allow it to run at speeds reaching 2.0 m/s, and jump to a vertical height of 0.48m (Ghost Robotics, 2016).

As showcased by the previous examples, actuation is an important facet of robot design. With new actuators being created and tested, such as shape memory alloy, piezoelectric, or magnetostrictive actuators, the limits are being pushed past the traditional capabilities of robotic actuation (Hollerbach, 1992).

2.4.4. Computation

The field of computation has taken major strides in recent years, led in part by the miniaturization of processors and computers for consumer mobile devices. Manufacturers such as Nvidia and Intel are producing devices smaller than ever before, with greater computational power and lower electrical consumption than previously available (Shaller, 1997).

The Ghost Minitaur, mentioned previously, uses onboard a Raspberry Pi 3 mini-computer (Ghost Robotics, 2016). This system-on-a-chip is a lightweight device that offers similar functionality to a full computer, for extremely low cost, and an equally small form factor. While the Raspberry Pi trades size and cost for computational power, more expensive (and more powerful) solutions exist to fill the market for high performance needs.

2.5. Current State of the Art

In the years since 1969, with General Electric's Walking Truck, technological advances have led to the creation of highly sophisticated and specialized robots to aid humans with a multitude of tasks (Raibert, 1986). Three such specialized robots include Boston Dynamics' Spot, MIT's Cheetah, and Ghost Robotics' Minitaur. Additionally, legged robots provide an excellent platform for subsumption architecture developed by Rodney Brooks in 1967.

Figure 7: Boston Dynamics SpotMini

Boston Dynamics' Spot was built to develop from its predecessor, LS3. Spot was built in a partnership between Boston Dynamics and the US Defense Advanced Research Projects Agency to assist the US Marine Corps with missions. Although Spot must rely on an operator for instruction, it was designed to navigate terrains such as hills, woodlands, and urban areas in order to assist Marines via scouting possibly dangerous areas as well as carrying small items alongside Marines. SpotMini, the most recent iteration of Spot, is a 55lb robot that has a 5DOF arm attached to the front of its body which contains a camera (Boston Dynamics, 2016). Unlike its predecessors, SpotMini does not include any hydraulic actuation so its movements are quieter and could feasibly assist US Marines in risky stealth operations (Ackerman, 2016).

Figure 8: MIT Cheetah 2

Another such robot is the MIT Cheetah II, which was designed as a platform to study dynamic locomotion in quadrupedal robots while modeling a cheetah. Not only can this robot run at speeds of 6m/s, but it can also automatically detect and avoid objects up to 80% of its leg length by jumping over them. (Bosworth, 2015). The object detection is done by a 2D distance sensor mounted at the front of the robot which determines both how far and how tall the closest object is. The robot then readjusts its step placement in order to leap over the object while not hitting it. At lower speeds, the Cheetah can navigate through uneven terrain.

Figure 9: Ghost Robotics Minitaur

The Minitaur, developed at the University of Pennsylvania in collaboration with Ghost Robotics, is a quadruped robot with symmetrically driven 5-bar linkages used as directly driven legs, meaning there are no gears or other mechanisms between the legs' drive motors and the legs themselves. (Kenneally, De, and Koditschek, 2016). This leads to an increased power efficiency as compared to motors with gears which may only have a maximum of 90% efficiency. This lead to the Minitaur being able to bound at a speed of 1.5m/s and jump about 50 cm vertically. The precise nature of the Minitaur allows for control across a variety of terrains while also being able to make small and precise movements.

Biomimicry is becoming more common in robotics. It's an obvious source of insight, delegating design decisions to thousands of years of evolution (Haberland, 2015). It is worth noting, however, that while inspiration can come from nature, application specific decisions must still be made to optimize the solution.

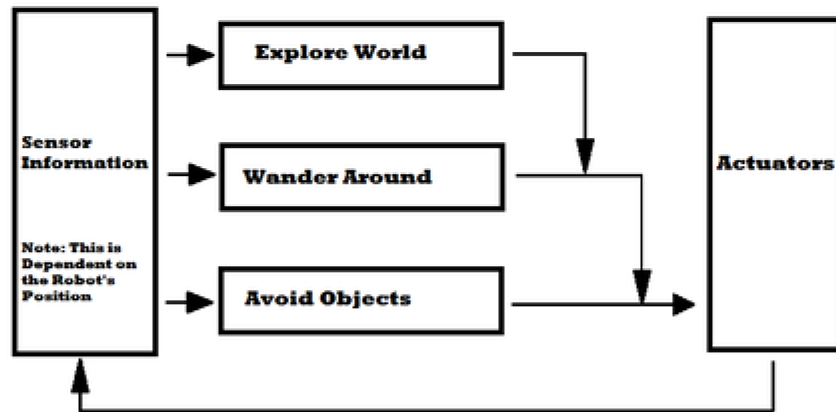


Figure 10: An example application of subsumption architecture

In 1967 Rodney Brooks developed a software architecture called subsumption which simplifies a system to layers of finite state machines. Depending on the state of the robot and its surroundings, this architecture would help choose which action is the most important to act upon. One application of this architecture was a robot developed by Brooks called Allen. The first layer of this robot was to avoid obstacles. If approached, the robot would move to avoid the oncoming object. Otherwise it would stay in place. The second layer was to have the robot wander around its surroundings and decide a new direction every 10 seconds. The third and final layer of this robot was to use its sonar sensor to detect far away objects and move towards them. Using this architecture allows a robot to give priority to different tasks depending on its surroundings, allowing an alternative application to modeling animal behaviors than traditional AI systems. Upon adding more layers, a system becomes more complex in its ability to react to its environment (Brooks, 1991).

3. Project Strategy -- Design Specification

3.1. Hardware Design

The hardware design is focused around building a device that is inexpensive to maintain, flexible enough that it has room to grow, and able to “Act” (actuate the system properly). By prioritizing parts that can be manufactured with rapid-prototyping machines and commercial off the shelf components for computation and control, we proposed a system that could be built and programmed by anyone with minimal application-specific knowledge.

3.1.1. Material

The robot's design with regards to material selection can be divided into three component classifications: moving components, ‘stationary’ components, and springs/dampers. Each classification has different requirements for general material properties, structural properties, and mechanical properties. These requirements are defined as follows:

1. **General Material Properties:** The material must be cost effective against its structural and mechanical properties as well as against its own properties. (e.g. Material Cost vs. Density, Material Cost vs. Max Stress, Material Cost vs. Manufacturing Cost)
2. **Structural Properties:** These are properties that define the design of the structure. (e.g. for 3D printed parts structure geometry and fill percentage dramatically affect the strength of the part.)
3. **Mechanical Properties:** These properties define the material itself. (e.g. Young’s Modulus, Shear Strength, Density, Thermal Expansion, Melting Point)

One goal for this project was to make our robot cost effective, using low-cost, yet high quality, materials to achieve high-performance motion. To achieve this, our design must have a relatively low weight, yet have the strength to withstand forces exerted during motion. We found the most reasonable materials to be 3D printed PLA and aluminum. These materials have comparable densities and strengths, depending on the in-fill of

the 3D printed part. Their material costs were also relatively low, and they were easy to manufacture. Our selection for materials based on the component classifications can be found below.

1. Moving components: These components include the legs and gearing. The legs were made almost solely out of PLA with a mid-to-high range in-fill density and a honeycomb structure. The gearing system was tested with plastic VEX gears.
2. Stationary Components: These components comprise the body of the robot. This includes the actuator nacelles, electrical component housings, and the main torso. These parts were 3D printed to keep the system lightweight and because there are no extreme forces acting on the body. It also decreased manufacturing costs.
3. Springs/dampers: The springs added to the design were used for retracting the leg to its initial position. Damping was done by adding 'soles' to the bottom of the feet. This helped reduce the forces on the PLA and reduced the chance of material failure. These 'soles' were rubber no-slip shoe pads.

Along with these three subsets of components were miscellaneous fasteners. These were mostly common stainless steel fasteners either from typical VEX hardware kits or from a hardware store.

3.1.2. Actuation

Actuation of the legs occurs at two locations, as shown in Figure 9: at the shoulder joint and just below the elbow joint. Actuation at the shoulder was the result of a simple electric DC motor. Our torque analysis in Section 3.3.2 indicated that a VEX 393 motor has the capability to actuate the lower leg, and further tests conducted verified this claim. The secondary actuator is a servo mounted on the body and connected to the lower leg via cable system. As the position of the servo is changed, it turns a cam affixed to the output shaft, putting tension on the tendon cable. The tendon is part of a series elastic system, with a tensioning elastic tendon providing force opposite the servo.

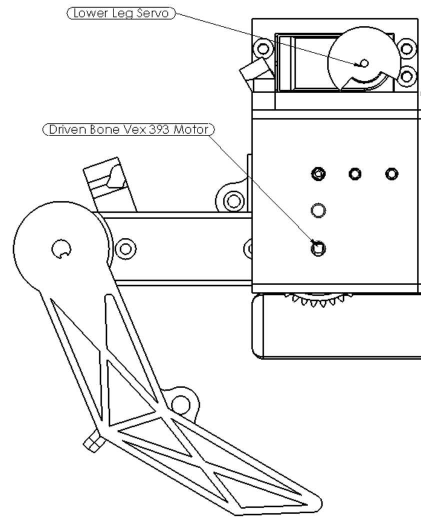


Figure 11: Leg Actuator Locations

High torque servos for hobbyist use exist that can provide the necessary torque, as determined using our dynamic equations, to actuate such a tendon system. One such type designed for use in sailboats is known as an RC Winch. Able to provide up to 1.7N/m of torque and 180 degrees of rotation, significant redesign here can be a failsafe for the lower limb control.

3.1.3. Electrical Design

Another part of the “Act” division of the system, the electrical design distributes power to the different sections of the robot. Included in the electrical design is a metaphorical bridge between the “Think” and the “Act” in the embedded controller that communicates with the onboard computer. The electrical system is the connection between the controls system and the actuation of the robot.

Power Distribution

The electrical subsystem is comprised of separate parts: Actuator power, Controller power, and Battery. While a single battery system will provide power for the entire system, it is possible that the control systems will require a different provided voltage than the driver circuits of the motors. Accommodating supply voltages for each component in the system is important to maintain proper operation of each device; but can be done independently of any battery.

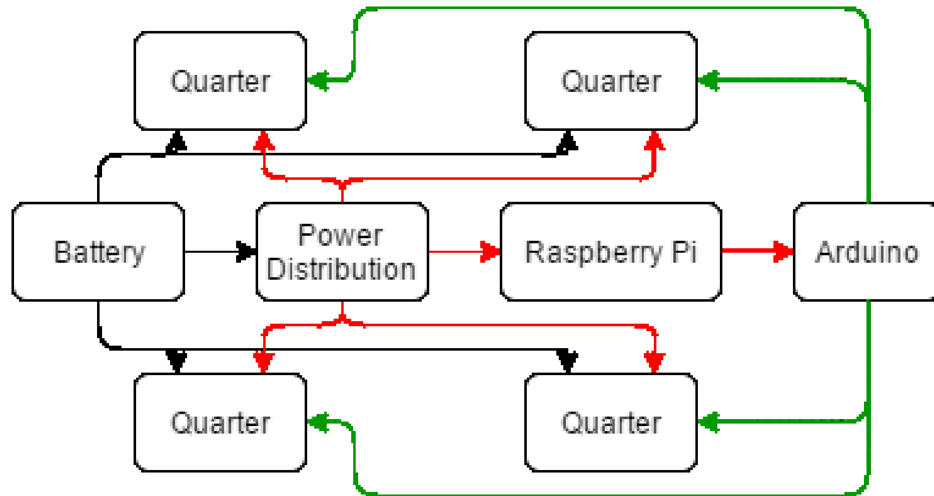


Figure 12: Electrical Diagram showing Actuator Power (Black), Controller Power (Red), and Signal (Green)

The battery has five important attributes on which it should be selected: Operating Voltage, Energy capacity, discharge rate, weight, and price. The weight of these factors varies, and each has its own importance. Most important across all of them, however, is availability; the battery must be available both in availability to ship, and price to purchase. In preliminary research, available batteries typically had the following characteristics:

- Voltage: 7.4V->11.1V
- Capacity: 4Ah->7Ah
- Discharge: 10c->90c
- Weight: 350g
- Price: $25->$65

Lithium Polymer batteries are a well-established technology with a strong backing of investment and use in the hobbyist market. Better performing than NimH cells, and weighing less, these cells will work well for our application. Calculating the discharge rating we require from the battery pack is approximated easily with the current draw of our system. Approximating maximum current draw from our system:

$$\Sigma(power) = \Sigma power(motors) + \Sigma power(servos) + \Sigma power(computation)$$

Power draw for computation is expected to remain below 15W, the main draw on the

battery will come from the servos and the motors. Peak power rating for the motors is listed as 3 Amps at 7.2V, or 21.6W per motor, peak rating for the servos is 10W. Assuming full peak power consumption, then, the following can be said:

$$\Sigma power(servos) = 40W$$
$$\Sigma power(motors) = 86.4W$$
$$\Sigma power(computation) <= 15W$$
$$\Sigma(power) = 141.4\text{W}$$

Rated discharge capacity is the maximum current draw allowable from the battery, as a factor of its ‘c-rate’ discharge capacity. For instance, for a battery rated at 5.2Ah and 10c, the maximum current discharge is 52 Amps. From the above calculation of theoretical peak power draw from the platform we can calculate the maximum operational current draw from an 11.1V battery would be approximately:

$$\Sigma(power) \text{ / Voltage} = 141.4\text{W / } 11.1\text{V} = 12.1\text{A}$$

This indicates that a 10c battery is within the minimum requirements for the current specified actuation, and leaves headroom for further modifications to the specification.

Circuit Design

Other important considerations that are left to the specifics of the design, are that the layout of the electrical system should emphasize cleanliness of cable routing, and modularity of the segments of the robot. Disassembling and reassembling the robot will be a common action, and coupling between layers should be low.
The low voltage power distribution will be managed primarily by the circuitry of the robot. Using a DC-DC converter to step down the voltage from the 11v of the battery and then distributing the low-voltage. The current on this bus will be limited to primarily logic-level distribution, so the primary concern will be to design for modularity.

Sensor Package

In order to make the robot both modular in design and flexible in application, many different sensors were considered. These sensors were broken up into two categories; sensors required for basic motion and sensors required for more advanced functionality. The sensors available for each category are as follows:

- Movement
    - Encoders - Encoders need to be attached to the motors in order to

determine the angle of the motor. Knowing the angle that the motor is currently at allows the software to know the position of the robot and all of its limbs. From this, it can plan its next step and execute its movements properly and accurately.

  - Inertial Measurement Unit (IMU) - An IMU can be used onboard the robot to determine the rotation of the robot. From the IMU, the tilt of the robot can be determined, and can enact any movements needed to prevent the robot from tipping over. In addition, the IMU can provide useful information regarding velocity that can be used when different gaits need to be executed.
- Advanced Functionality
  - Camera - In order to run SLAM, a camera capable of producing 3D data is needed. While there are many options when it comes to camera, for the purpose of this project, an RGB camera is recommended due to their low cost vs. overall performance.
    - By having an RGB camera, it opens up other possible avenues of Human-Robot Interaction to explore with this platform, such as face detection and object recognition.
  - Motion Sensors - While an RGB camera such as the Kinect or Xtion Pro provide infrared information, it may also be beneficial to have additional infrared motion sensors on other parts of the platform to enable more functionality such as further object detection and security features.
  - Audio Inputs - By having audio inputs, the robot can be programmed to respond to various voice commands that can enhance the interaction between the user and the robot.
  - Bump/Touch Sensors - Bump/Touch sensors can also be integrated to further engage the end user into interacting with the robot. Bump sensors on the body or the head can be used to represent a form of “petting”.
  - LCD/LED display - While not technically a sensor, an LCD/LED display can be added to the robot in order to relay information about the current state of the robot. This information can pertain to the robot’s functionality, or can be used to depict the robot’s level of interaction with the user and portray moods, much like the Sony AIBO.

3.2. Control System Design

Our specification calls for a divided control system, separating higher level command into an onboard computer and with motion and hardware control relegated to a coprocessor board and interface. This design distributes the required computation, preventing any single processor from being overtaxed. Each system performs the tasks that it is suited for and relays information to the other systems when done. The higher level system performs the “Think” aspect of robotics, deciding how the robot should act based on its exteroceptive sensors (the vision system and human interface). The lower level embedded system then bridges the “Think” and the “Act,” issuing electrical instructions to the actuators.

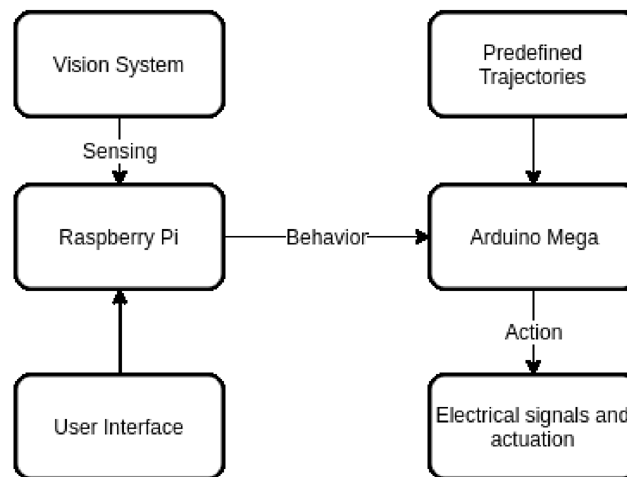


Figure 13: Control System Architecture

Embedded Controls System

Robotic systems require a motion controller to interface higher level systems with the real world. This controller has to maintain real-time processing during motion execution in order to ensure smooth operation of the robot. Real-time processing in this case is defined as the microcontroller completing all previous calculations before the next interrupt or time slice (sample time) occurs.

To achieve this, two limitations are defined:

1. The number of instructions to be executed must be small.
2. The clock rate must be fast enough to accommodate the instructions that must be executed.

Another consideration in choosing a microcontroller is the ease of integration to the system. System modularity can be improved by using commercially available controller boards.

Input and Behavior Controls System

Controlling the onboard camera, and processing its input is computationally expensive. Typically, higher level computation, such as computer vision and SLAM algorithms require significant computing resources to operate at the rate required for robot operation.

In order to implement the desired behaviors, the robot itself will operate the low cost but still reasonably powerful Raspberry Pi 3 to manage exteroception (the vision system and user input). The Raspberry Pi will process the relevant data and perform the necessary computation and then communicate with the embedded controller to perform actions with the robot. By separating the computation and adding a low cost computer, the robot is not limited by its hardware, thus allowing for a greater range of applications to be developed.

3.3. System Analysis

To ensure that the system operates correctly, analyses were completed to evaluate various parameters involved with movement, namely: kinematics and kinetics analyses. These analyses provided the information necessary to determine the positions, velocities, and forces for the robotic limbs. These computations were completed with a focus on the legs, using the shoulder joint as a ground link (Figure 13). Using this assumption, the legs could undergo the calculations for a modified 2-DOF planar arm manipulator. Any values mentioned in this section are estimated and the results are representative of the system at a static state.

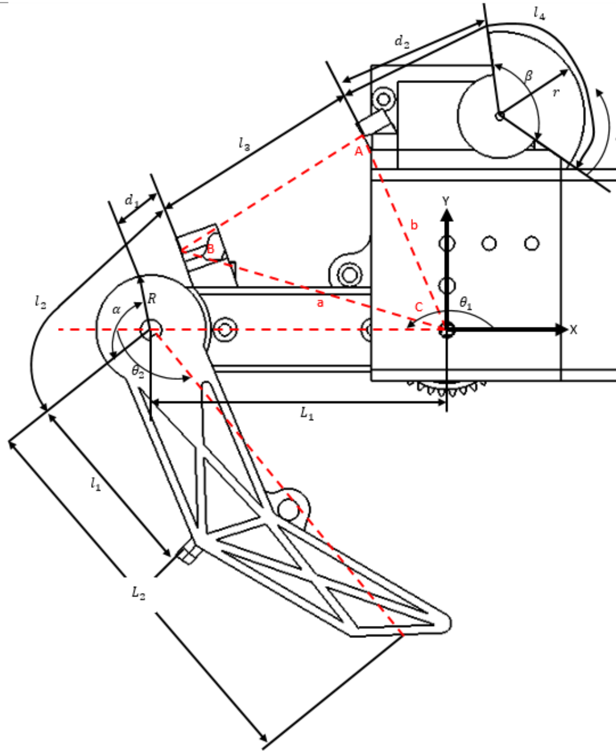


Figure 14: Working Model for Robotic Leg Kinematics

3.3.1. Kinematics

To describe the positions of our robot's leg links, forward kinematics were used. Because our leg design is not that of a simple 2 DOF planar mechanism, custom equations needed to be developed. These equations were focused on determining the position of the lower leg utilizing the rotation of the upper limb and the servo's rotation as inputs. The process for determining these positions included, first, creating the variable model for evaluation shown in Figure 13. Next, equations of planar motion were applied for the given variables.

$$x = L_1 \cos(\theta_1) + L_2 \cos(\theta_1 + \theta_2)$$

$$y = L_1 \sin(\theta_1) + L_2 \sin(\theta_1 + \theta_2)$$

Then, the equations were developed to include the change θ_2 as the servo was rotated. This involved determining the length of the tendon cable and relating its change in length to a change in angle at the knee joint. The cable length was found using the following equations.

$$l_1 = const.$$

$$l_2 = d_1 + \frac{\alpha \pi R}{180}$$

$$l_3 = \sqrt{a^2 + b^2 - 2ab\cos(C + \Delta\theta_1)}$$

$$l_4 = d_2 + (\beta - \varphi)\frac{\pi r}{180}$$

$$l_{cable} = l_1 + l_2 + l_3 + l_4$$

The change in cable length was then changed into a change in angle around the knee which, in turn, specifies a change in angle of the lower leg. This was included in the kinematic equations and resulted in the following final equations.

$$x = L_1 \cos\theta_1 + L_2 \cos\left[\theta_1 + \left[\theta_2 - \left[\alpha - \frac{[(l_{cable} - l_1 - l_3 - l_4) - d_1]180°}{\pi R}\right]\right]\right]$$

$$y = L_1 \sin\theta_1 + L_2 \sin\left[\theta_1 + \left[\theta_2 - \left[\alpha - \frac{[(l_{cable} - l_1 - l_3 - l_4) - d_1]180°}{\pi R}\right]\right]\right]$$

These equations were verified to be accurate to within 0.125 in. These equations allow for various angles to be tested to determine end-effector positions. For example, using ▯$_1$=3.50 in, ▯$_2$=4.6352 in, ▯$_1$=195 deg, φ=15 deg results in an end-effector position of (x, y)=(-2.89 in,-5.515 in).

3.3.2. Dynamics

The dynamic analysis involved applying the equations for a simple two degree-of-freedom robotic arm using Lagrangian mechanics. This evaluation was necessary to ensure that the actuators will have enough torque to move the limb, and also so we can ensure that the joints do not tear themselves apart. The system was modeled as shown in Figure 13. The assumptions were made that the two links have distributed masses and the center of mass of each link is at the exact center of each link. Furthermore, the links were modeled as a link rotating about a fixed axis and a link rotating about a moving axis.

After solving the kinematic end effector positions, equations for torque in the two limbs were obtained by applying the kinematic equations to the Lagrangian energy equation.

The derivatives of the Lagrangian were taken and resulted in the following torque equations.

$$T_1 = \left(\frac{1}{3}m_1L_1^2 + m_2L_1^2 + \frac{1}{3}m_2L_2^2 + m_2L_1L_2C_2\right)\ddot{\theta_1} + \left(\frac{1}{3}m_2L_2^2 + \frac{1}{2}m_2L_1L_2C_2\right)\ddot{\theta_2} - (m_2L_1L_2S_2)\dot{\theta}_1\dot{\theta}_2$$
$$- \left(\frac{1}{2}m_2L_1L_2S_2\right)\dot{\theta_2}^2 + \left(\frac{1}{2}m_1 + m_2\right)gL_1C_1 + \frac{1}{2}m_2gL_2C_{12}$$

$$T_2 = \left(\frac{1}{3}m_2L_2^2 + \frac{1}{2}m_2L_1L_2C_2\right)\ddot{\theta_1} + \left(\frac{1}{3}m_2L_2^2\right)\ddot{\theta_2} + \left(\frac{1}{2}m_2L_1L_2S_2\right)\dot{\theta_1}^2 + \frac{1}{2}m_2gL_2C_{12}$$

where

$$\theta_2 = \theta_{2,original} - \left[\alpha - \frac{[(l_{cable} - l_1 - l_3 - l_4) - d_1]180deg}{\pi R}\right]$$

$$\dot{\theta_2} = \frac{-ab\sin(C + \Delta\theta_1)\dot{\theta_1}}{l_3} - \dot{\varphi}\left(\frac{r}{R}\right)$$

$$\ddot{\theta_2} = -\ddot{\theta}_1\frac{ab\sin(C + \Delta\theta_1)\,\dot{\theta_1}}{l_3} - \dot{\theta_1}\frac{ab\cos(C + \Delta\theta_1)\,\dot{\theta_1}}{l_3} + \dot{\theta_1}\frac{ab\sin(C + \Delta\theta_1)\,\dot{\theta_1}}{2\sqrt[2]{(a^2 + b^2 - 2ab\cos(C + \Delta\theta_1))^3}}$$
$$- \ddot{\varphi}\left(\frac{r}{R}\right)$$

and

$$C_1 = \cos(\theta_1)$$

$$C_{12} = \cos(\theta_1 + \theta_2)$$

$$C_2 = \sin(\theta_1)\sin(\theta_1 + \theta_2) + \cos(\theta_1)\cos(\theta_1 + \theta_2)$$

$$S_2 = \dot{\theta_2}\cos(\theta_1)\sin(\theta_1 + \theta_2) + \dot{\theta_2}\sin(\theta_1)\cos(\theta_1 + \theta_2)$$

.

From these equations, we determined the torques necessary to actuate the limbs, and identified and applied servos and motors which met these torque specifications.

4. Specific Application Parameters

4.1. Controls Processing

Managing the motion controls of the robot will require an embedded processor able to maintain real-time computation under our required load that is also able to analog and digital I/O. For our purposes, an Arduino Mega will provide sufficient capacity in both regards.

This Arduino device has onboard 54 Digital I/O pins, 12 Analog inputs, 2 Analog outputs, and an operating voltage of 7-12v. Current designs for the platform include minimum requirements for 10 digital inputs, 8 digital outputs, two analog inputs, and 1 analog output (speaker). Using the Arduino Mega leaves plenty of General Purpose I/O (GPIO) pins for expansion of sensors and controls.

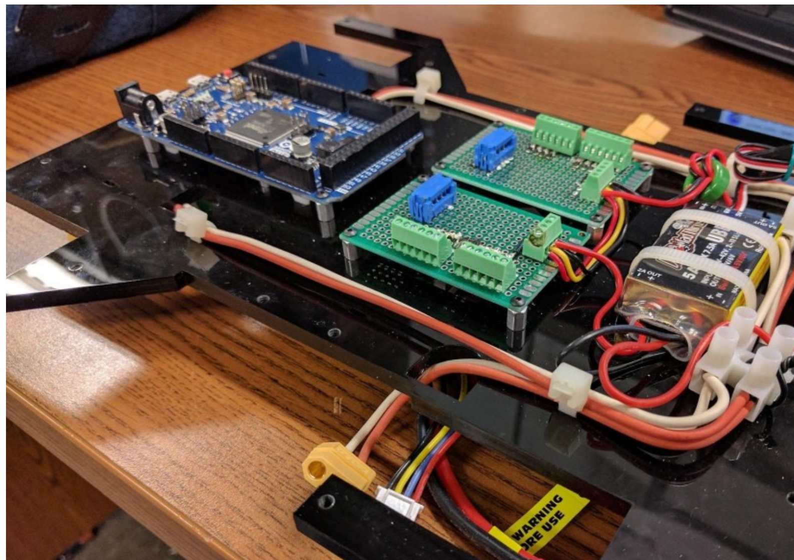


Figure 15: Electrical Subsystem layout

Discussion has been raised about the fact that the Arduino operating system is not a "hard real-time" system. This brings concerns of the ability of the device to process all of the required calculations and handle all interrupts inside of a given time division. In theory, the time required for a processor to execute a computation can be calculated from the number of instructions required for a given architecture to complete it and the

rate at which those instructions are handled. In practice, it is much simpler to time the execution of a given set of computations. Using the GPIO to indicate the start and end of computation, it was found that the Arduino Mega is capable of handling the required calculations in real-time.

4.2. Electrical Subsystem Design

As our main energy source, we will look to the hobbyist market for an inexpensive battery. Our team debated between the factors discussed in 3.1.2 and selected a Lithium Polymer battery pack rated for 5.2Ah at 11.1V. The specific battery is not the highest rated of those from our initial assessment, but was selected instead for its lower price and standard form factor. For instance its energy storage of 5.2Ah and discharge rate of 10C is outpaced by similar mass battery packs of 7.5Ah and 90C; our purposes won't require 90C of discharge, however, and at less than half the price, we can purchase multiple to obtain the same run time. Choosing a mid-grade battery will afford us the performance we need at a price we can manage.

Providing the required operating voltage for all of the components on the platform is dependent on the voltage of the battery selected. The most pressing constraint being the operating voltage of the Raspberry Pi onboard the robot. The Raspberry Pi has an operating input voltage of 5v. Voltages outside this range would have to be converted to suit. Given the battery specified above, a DC-DC converter is required to provide the computer with the correct voltage.

The motors we've specified, the Vex 393, have a rated operating voltage of 7.2V. In our experimentation, they were found to be resilient to an input voltage of 11.1V.

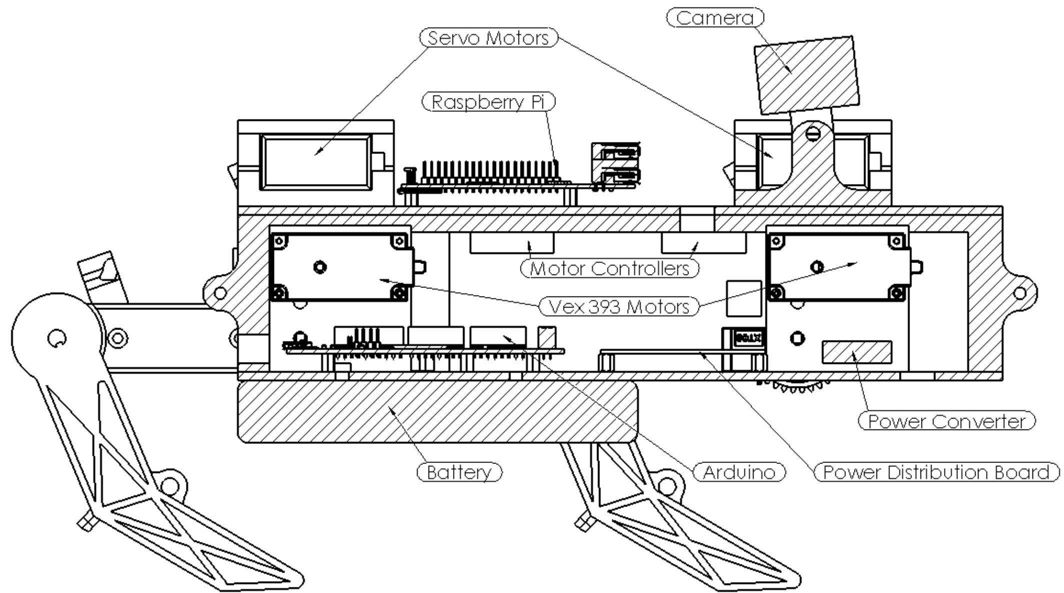


Figure 16: Robot Subsystems

The robot subsystems are divided among the top, inside, and bottom of the torso as shown in Figure 15. The battery is located beneath the device, motors, motor controllers, and the Arduino are located inside, and RaspberryPi and camera are on the top layer. Wiring onboard the device will be divided appropriately from the inner level to the outside to maintain a direct and clean distribution. Connectors between levels is important, to aid in the modularity of the system during maintenance, disconnecting all of the connections at the same time is an important feature. Furthermore, heat dissipation in the components will have to be monitored to ensure that overheating does not occur, especially in a heat-intolerant structure (ABS/PLA plastic). Our goal will be to make the sections as decoupled as possible, making maintenance easier and a better device.

4.3. Hardware Architecture Design

In order to decrease computational load and increase the modularity of the robot, the system hardware will comprise of 3 different controllers. Due to price constraints, it was decided that the system will be comprised of a RaspberryPi 3 as the main computer controlling the robot, its sensors, and its high level functionality, an Arduino Mega to control the motion of the robot, and a base station that will compute the more computationally intensive aspects of the robot. Each controller and the sensors they manage is described herein.

4.3.1. RaspberryPi 3

The Raspberry Pi 3 will act as the main computer for the robot. Between the Arduino and the Pi, it has the better processor, clocks at a faster speed, and supports a series of peripherals and libraries that will be needed, thus making it the better choice. The main sensor that the Raspberry Pi will manage is the camera. Since we will process SLAM onboard the robot, the robot will have perform the SLAM algorithm and any other high level planning on the Raspberry Pi.

4.3.2. Arduino Mega

The Arduino Mega will manage all sensors and hardware associated with motion. The Mega will manage the following items:

- Potentiometers
- VEX Encoders
- VEX Motors
- Servos

One complication we foresee is complexity in the application of the Vex Encoder Modules. Open source libraries for communication with these encoders are available, and will make their integration easy to do. As stated before, the Arduino Mega has enough computational power to process the sensors and controllers that it will manage.

4.3.3. Base Station

One possibility would be to operate a base station to manage the most computationally expensive programs. This would reduce the burden of the onboard computer, and increase the total availability of computational power.

The main functionality of the base station would be to run SLAM and any interfaces that will be needed to test and run the robot. During operation, the robot will transmit all of its camera information to the base station. The base station will then run SLAM and any other algorithms needed, and then transmit this information back to the robot.

Although SLAM algorithms do differ in execution requirements, the base station requires at least 4GB of RAM, a modern processor and be running a version of Linux compatible with ROS. It also requires wireless capability to communicate over Wi-Fi or Bluetooth to the robot's main computer. Many mid to high-range laptops will satisfy this need, and most desktop PC's that can be found at WPI are also able to fulfill the hardware

requirements.

Our application, however, will operate without a base station due to cost restraints and the ability of the Raspberry Pi to handle the processing we require.

4.4. Software Architecture Design

4.4.1. Software Flow

In Section 4.3, we discussed the various hardware systems that the robot will include. Each system will run its own software and execute its tasks based off of shared information. The intended software flow is as follows. The robot will be turned on, and will begin to take in data from its camera sensor.

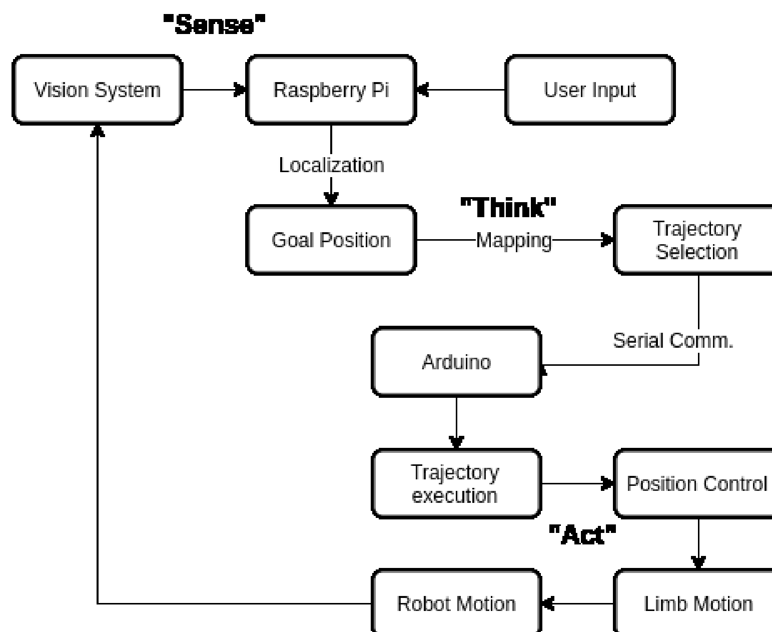


Figure 17: Robot State Diagram

The Pi will then process this information and transmit distances, orientations, headings, etc. to the Arduino Mega. Since the Mega's software will control all of the various motors and encoders, it will then actuate the robot accordingly. The Raspberry Pi will actively process input, so if an obstacle approaches or there is some disturbance, the robot will be able to replan and react immediately.

4.4.2. Operating System

To make the robot modular, flexible, and accessible, it was decided that we would use ROS as the primary operating system. Utilizing ROS creates a familiar environment for software development for future work on the system. It also makes it easy to integrate software written by other software developers, including implementations of SLAM and user interface. ROS also trivializes communication between different platforms on the system, abstracting network communication. Using ROS also opens the system to future integrations of offboard computation and hardware modification.

4.4.3. SLAM

One of the desired application goals for the robot is the ability for it to avoid obstacles and navigate around its environment. In order to do this, the robot must run a computer vision system and an algorithm that allows the robot to localize itself in its environment. In order to do this, we plan on using the SLAM algorithm. There are various considerations that need to be reviewed when selecting a SLAM algorithm to use. While many are the same, some differ in overall performance. Specifically, some SLAM algorithms work better than others when there are height or rotation variances between frames. Some algorithms such as RGBSLAM are able to continue operating correctly despite these variances, while other SLAM implementations yield highly inaccurate results. Various SLAM algorithms will be implemented and tested in order to fully understand the mapping capabilities of the robot.

4.4.4. Image Correction

Due to the movement of the robot, and how it may be angled to one side or the other depending on the number of points of contact, the images obtained from the camera may need to be filtered and manipulated in order to make the SLAM algorithm work properly. The images will be filtered using data from an IMU. The IMU will provide the program with the orientation of the robot. The images will then be offset to adjust for the change in orientation, thus giving us accurate images to use. This is important because if the images are skewed, then the SLAM algorithm will think that the features that were extracted have moved or are further away than they are. If that happens, then the robot will not be able to properly localize itself and may end up in a “locked” state where it does not have any idea where it is at. Implementation of the image correction

can be found in Section 5.6.

4.5. Gait

Walking is done by actuating the legs in order and position to form a gait. Our design calls for two types of forward motion: static walking, and dynamic running.

Inspiration for these motions will come from the investigation of animal gait. Investigation into the gait of moving animals has been recorded since 1870, when Eadweard Muybridge first photographed the gait of a horse (Williams, 1992). We hope to mimic the motion of animal gait in our robot for our dynamic and static walking. An important note here is that the current design of the legs is such that a full stepping motion as found in many living quadrupeds will likely be impossible; this design was chosen regardless because the added complexity to adding an additional degree of freedom would be restrictive in cost and design. Accounting for this, however, will require innovative planning in the actuation of the upper limb.

Dynamic walking will likely be able to draw more closely from biology. Likely not in direct motion mimicry, but rather through a comparison in the timing of the legs. Equestrian gaits are grouped based on speed and stability, some having been trained into horses by humans (Harris, 1993). Our initial goal is to reproduce two gaits with the quadruped: Trot and Galloping. Further experimentation could produce results in other gaits, but starting with these two will provide a base to demonstrate the ability of our platform.

4.6. Behavioral Control

Behavior control is the thematic integration of “Think” and “Act.” Based on a given input, the robot’s main controller, in this case the Raspberry Pi, will decide the action it wishes the robot to produce. The Raspberry Pi then will issue movement commands (distance and direction controls) to the embedded controller onboard the platform. While the Arduino Mega will control the specifics of motion on the robot, the Raspberry Pi will determine through its defined criteria which of those motions should be executed.

User interface will be managed through a mobile application programmed for a smartphone. The smartphone is connected to the Raspberry Pi through a wireless protocol such as Bluetooth, allowing for an interface between the person and high level

control of the robot such as position, gaits, speed. Having this interface will permit control of the robot and dynamic changes of its behavior during tests. Additionally, images from the on-board Xtion Pro camera can be streamed to the user interface so the operator would not need to be in the same room as the robot, but could still gather information of the robot's surroundings.

4.7. Bill of Materials

The final step for our team was generating a bill of materials. This list specifies the components necessary for the construction of the system from actuators, to sensors, and to the miscellaneous nuts and bolts. Our resulting bill of materials is shown in Table 1.

Table 1: Bill of Materials

Component	Name	Quantity	Price (per unit)
PLA Filament	PLA Filament	1 kg	$22.99
Battery	Multistar High Capacity	1	$25.99
Motor	Vex 2-Wire 393	4	$14.99
Motor Encoder	Vex 393 Encoder	4	$15.00
Motor Controller	Vex 29	4	$10.00
Servo	Turnigy 1501	4	$15.94
Camera	ASUS Xtion Pro Live	1	$187.58
Computer	RaspberryPi	1	$35.70
Processor	Arduino Mega	1	$32.13
Assorted Hardware	COTS Hardware (nuts, bolts, pins, etc.)	1	$60.00
Total			**$588.11**

5. Application

5.1. Rapid Prototyping

5.1.1. Appropriate Clearance for Parts

To ensure that the interacting parts can move with little hindrance, proper clearance among parts is necessary. Furthermore, during rapid prototyping, the dimensions of a part can change due to the heating and cooling of the building material. For most interacting components, a clearance of roughly 0.1” was applied in the CAD models. This allowed for expansion or contraction of the material during production as well as provided room for possible lubricants or friction reducing components. A few parts, however, were designed to have a tight or even press fit, such as the bearings or the cam. This fit type was necessary when motion between interacting parts was not wanted such as slip between the cam and servo output gear.

5.1.2. Part Design for Rapid Prototyping

When designing the parts to be created using rapid prototyping, care was taken to ensure proper functionality between the mechanical, electrical, and computational aspects of the system. Originally, we designed the system to seemingly keep these three systems separate. However, this idea was quickly discarded, and a more modular, and integrated system was developed.

5.1.2.1. Original Design

Body

The original design for the body of our quadruped, as shown in Figure 17, was a multi-layered assembly whose layers consisted of a battery compartment, a drive bed, and a computation platform. While this design provided a rigid working platform, it made it difficult to swap out single parts in case of malfunction. It also drastically increased the production time by increasing the number of parts that needed to be made by rapid prototyping methods. Our final realization was that the body was too large to produce using any of the rapid prototyping methods easily accessible to us.

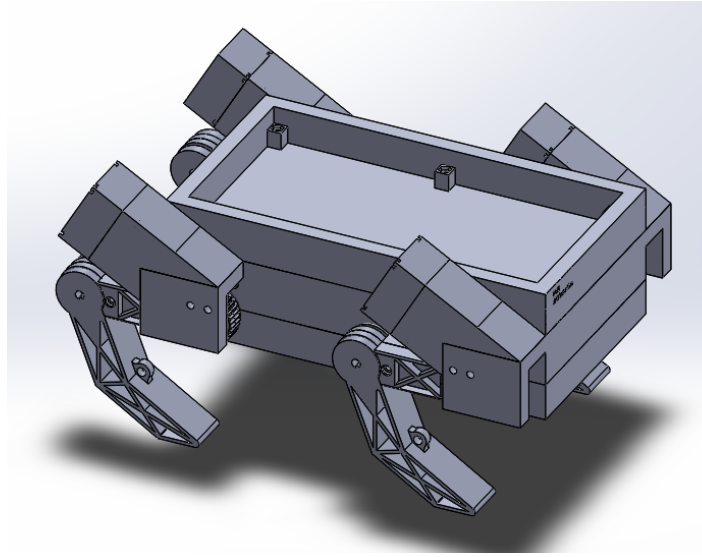


Figure 18: Original Robot Design

Legs

The original leg design, shown in Figure 18, used a two-piece driven 'bone' attached to a lower leg. These pieces featured trusses in their design to lower weight while retaining strength. The driven bone was fitted with two 36-tooth Vex gears which were actuated by a 12-tooth gear driven by a Vex motor. The lower leg had two attachment points for motion actuation. The first, seen in the figure below, was linked by a spring to the underside of the driven bone, as its function was to return the leg to its static position. The second point is in the back of the lower leg directly behind the first point. This attachment was for the cable which was to be actuated by a Servo. Areas of concern about this design were mainly due to friction between joint interfaces or where the cable for actuating the lower leg passes over the driven bone-lower leg joint.

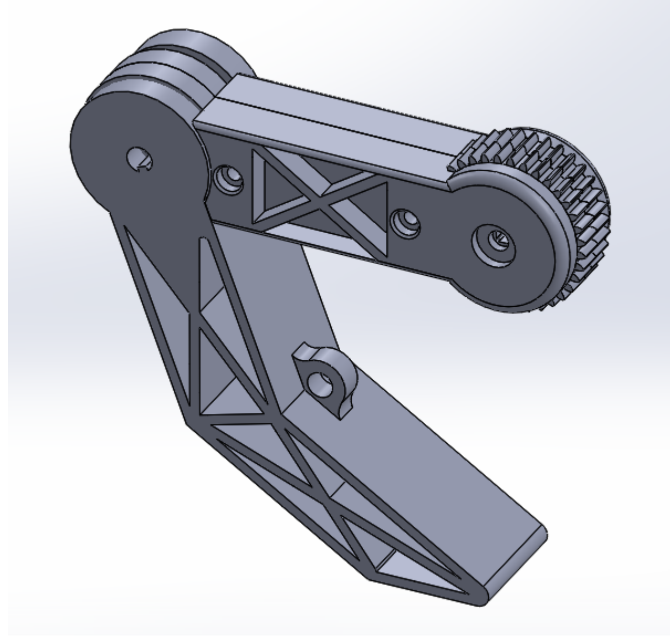


Figure 19: Original Leg Assembly

Nacelle

Our nacelle was originally designed as a housing for the servo which was being considered for lower leg actuation. The nacelle can be seen as the structure which leans toward the rear of the system in Figure 17. These designs were also quickly eliminated due to their lack of functionality. Their design would require a pulley to be in place to allow a cable to be properly attached to both the servo and the leg assembly.

5.1.2.2. Final Design

Our final system design, shown in Figure 19, was much more practical in its integration of our three subsystems. They were moved closer together as the main body was reduced to a single main torso to reduce production time, cost, and effort.

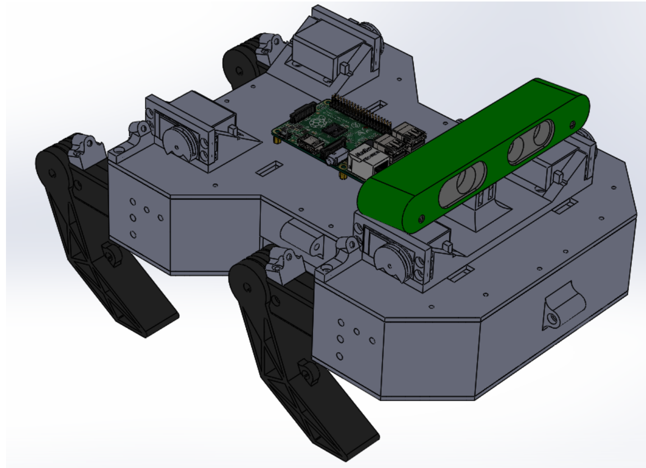


Figure 20: Final Robot Design

Body

The body was designed to increase the modularity of the system as well as improve access to components in case of malfunction. The three-layer system was scrapped and replaced with a single, main torso contained by two, thin, outer layers for support. The body was then split into four quarters, as shown in Figure 20, to make the system more modular and to allow for rapid prototyping methods to be implemented. The outer layers are pieces of high density fiberboard, which provide mounting platforms for components, as well as framing for connection of the four quarters. The fiberboard provides some flexibility, however its strength is expected to securely hold the body together.

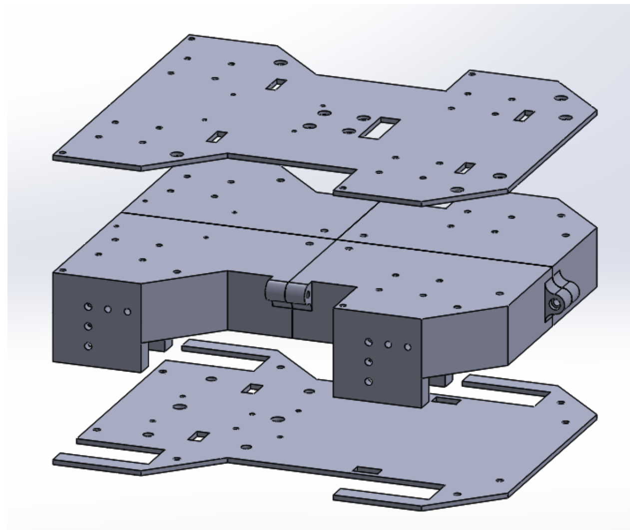


Figure 21: Final Body Assembly Design

Legs

The legs received very minimal changes to their design. Major modifications included adding a channel at the intersection of the driven bone halves to align the guiding sheath for the cable, removing the trusses from the driven bone, and adding a counterbore for a bearing to the driven bone. The final design can be seen in Figure 21.

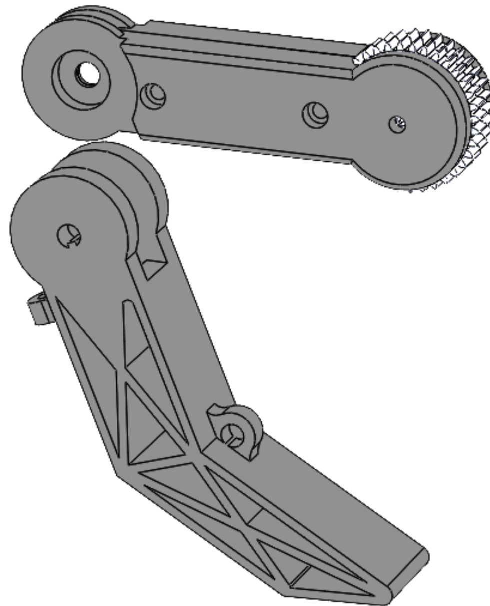


Figure 22: Final Leg Assembly

Nacelle and Cam

The lower leg system received the most dramatic alterations with the nacelle being designed to now hold a servo motor, and including a custom cam for leg actuation. The nacelle assembly is shown in Figure 22. The nacelle has a very simple design, with risers to securely hold the servo in place and an attachment point at which the guide sheath for the cable is terminated. The cam was designed to account for the change in cable length as the driven bone is rotated downward. As the driven bone rotates down, the cam will rotate toward the rear to provide the needed cable to prevent the lower leg from actuating. When the lower leg needs to be moved, the cam quickly reverses its direction to provide the necessary output motion.

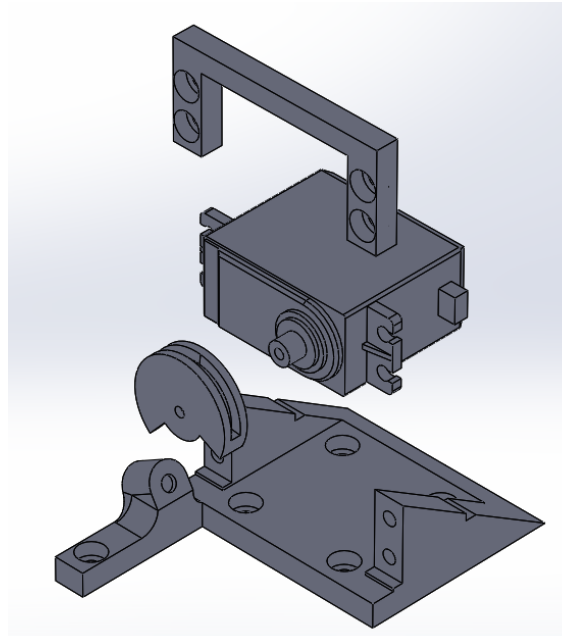


Figure 23: Nacelle, Servo, and Cam Assembly

The nacelle located at the front of the robot also serves as a mounting point for the on-board camera, as can be seen above in Figure 19.

5.2. Interfacing Rapid Prototyping and COTS

While the parts to be produced by rapid prototyping were designed with longevity in mind, stronger commercial-off-the-shelf (COTS) parts were required to reduce wear on the parts. These COTS parts were also required to keep the costs for attachment mechanisms and fasteners low.

Shafts and Bearings

Our designs are made to use commercial parts to increase the life of the system. This includes using connecting rods or shafts that reduce friction on the parts, and including bearings at the contact points for any rotating components. For example, the elbow joint at the intersection of the driven bone and lower leg includes a hole for a keyed shaft in the lower leg, and counterbored holes for rotary ball bearings in the driven bone. The keyed shaft prevents rotation of the lower leg about the shaft, while the bearing reduces friction at the joint. Similar configurations occur at the connection points for the gears, where common Vex flat bearings are used to hold the square shafts used in Vex models.

Cables and Fasteners

To join the various structural to moving components in the system, commercially available wire rope and fasteners were selected for use. Specifically, stainless steel wire rope was selected to connect the servo/cam to the lower leg. As for fasteners, hex head screws were selected as the main fastening components, varying in length and head type as required by their application. For example, at the end of the nacelle are holes that allow for a counterbore which can accommodate the head of a hex head cap screw, however when the head is to sit flush with the top of the hardboard frame, hex flat or rounded screws may be necessary.

5.3. Actuator Testing and Integration

5.3.1. Initial Test Results

Original design of the leg called for a dual-type actuator system, DC motor with encoder on the upper joint, and an electromagnetic solenoid actuating a tendon to the lower limb.

These designs were updated after initial testing showed that the motor on the top joint was effective for controlling the top of the system. It was surprisingly effective, in fact, given the limited precision in the encoders attached to the motors. Position control testing was affected using analog inputs on the microcontroller to read the position of a potentiometer that was then sent to the motor and leg system. This was used to tune the system’s PID loop as discussed in 5.3 below.

Figure 24: Leg testing configuration

5.3.2. Test Results and Redesign

The upper joint functioned effectively per the initial design. It was the lower joint of the leg that caused issues in our initial revisions of the hardware. The solenoid specified for control of the lower limb was simply unable to provide enough force to actuate the limb when extended to a meaningful distance. After a short period of testing, the solenoid was scrapped for a high torque servo connected to the tendon by a cam.

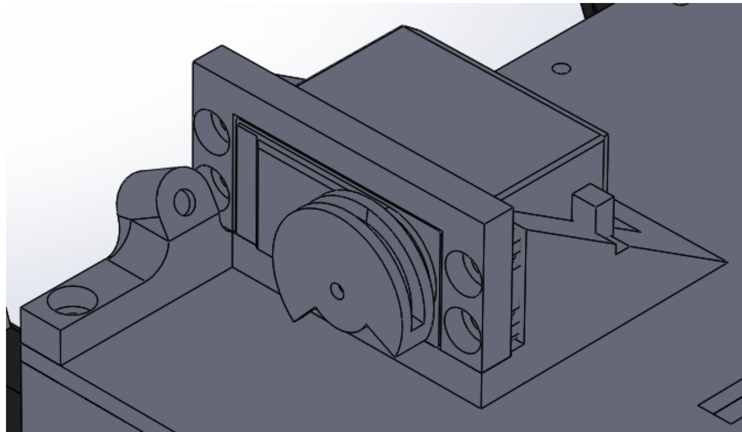


Figure 25: Redesigned Servo and Nacelle

A redesign resulted in the above design for affixing the servo to the main drive-bed assembly. Further testing with this assembly showed excellent results in torque and direct position control of the lower limb.

5.4. Controls Programming

Initial testing and construction of the controls software suite progressed with one quarter of the robot for testing and debugging purposes.

5.4.1. Application of Control System

Using the microcontrollers outlined in previous sections, initial testing progressed with the installation and application of basic position control of both joints on a single quarter of the robot. A single PID position control loop controlled the upper joint motor output, with input from the encoder system affixed to the motor shaft. The PID constants were adjusted to conform to the mechanical properties of the system, and tuned experimentally.

Control of the servo is done not through PWM as initially hypothesized, but rather through Pulse Position Modulation, changing the frequency of pulses instead of the duty cycle. Open source libraries exist to easily control such systems with the given microcontroller, and were implemented in our system to control the upper joint of the robot. This was integrated into the system due to the modular nature of the controls software; a change in only one place changed the lower limb control from binary to position based.

5.4.2. Trajectory

The codebase implements a trajectory control system that provides a datatype and execution parameters for position “frames” that are executed sequentially. These frames have parameters for positions for each joint on each limb, and the timing for its execution. To reduce execution time for the process, the trajectory relies heavily on memory pointers to reduce memory bus overhead retrieving data from storage.

In code, a frame is a datatype that is composed of an array of 8 integers. These integers represent the angles of the upper and lower motors/servos at a specific time slice. In order to execute a series of frames, a trajectory data type was created that contains a list of frames. Each trajectory object that is created represents a gait to execute. Since a trajectory can contain many frames, it is important to note the memory requirements for a trajectory. Each frame consists of 8 integers, so one frame takes up 16 bytes, therefore a trajectory that has more than 200 frames can take 3200 bytes of storage space. This becomes a problem when the Arduino Mega only has 8000 bytes of SRAM available to store all of the code. Exceeding the 8000 bytes available results in

unexpected crashes and errors when executing the code.

Because of their size, Trajectories had to be stored in Flash memory and read into SRAM only when needed during operation. The Arduino's architecture cannot abstract the memory hierarchy in the same way that traditional pointers do in a desktop programming environment. AVR provides function calls for reading from Flash memory during execution and instructing the compiler to store data in Flash memory (and not read it into SRAM at execution).

These functions are performed with the keywords *PROGMEM* in the type definition, and *read_pgm_near()* when reading the data from memory. Our control algorithm functions by buffering the next frame to execute before it is required to send the frame to the controllers, reducing the load on the I/O bus of the chip. In this way, only one frame is stored in SRAM at any given time, the rest being retrievable from Flash memory as required.

5.4.3. Gait Programming

To aid in the development of motions for the robot, a Matlab script was devised to output arrays of joint angles to be interpreted as trajectories. The Matlab script creates trajectories by interpolating between goal poses set by a user. It then outputs a moving graph of the motion that can be used by the programmer to refine the motion as desired.

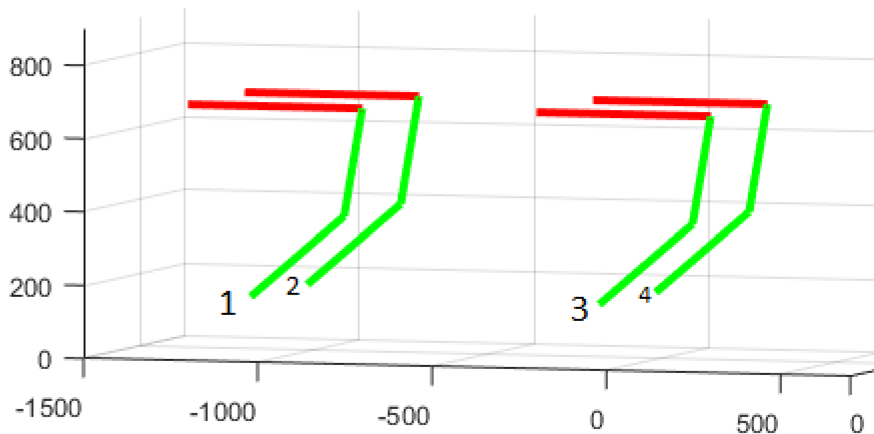


Figure 26: Gait planning graphic output.

Used to interpolate between joint positions, the script uses as input desired positions for each at a given state and the time between each state of the robot. The script then

generates trajectory data types for upload to the robot based on the states and times given by the user. The Matlab script was eventually refined to output header files that could be directly uploaded to the robot. The development of gaits can then be the simple process of refining the goal positions desired for the limbs and testing them on the robot.

5.5. Electrical Systems

The electrical system divides the robot into quarters and a central distribution board (*Figure 11: Electrical Diagram,* pg. 21). The quarters are connected to the central board in power and signal with multi-conductor cables and quick connectors. This allows the device to be disassembled or repaired with ease.

5.5.1. Electrical Design

Preliminary designs of the electrical system called for two power buses, one high voltage rail for primary motor output power, and a lower voltage source for signal and logic control. Component selection drove this design because the logic system of the Arduino and the motor controllers operates at a voltage much lower than battery output. Reducing the voltage of the lower bus was initially proposed using a DC-DC converter. Implementation, however, found that developing a converter circuits would be too expensive for use on this project. Instead, a commercial, off the shelf converter was acquired. Typically used for hobbyist applications, the switching converter provides a single circuit, rated for 7 Amps.

The 5 volt bus will be used to power both the control circuitry and the servo motors mounted on the nacelles. The servo motors are expected to produce electrical noise on this bus. To protect the control circuitry from damage due to this noise, capacitors will be added to the DC bus to clean the signal.

5.5.2. Wiring Design

The complex cabling required for the multicomponent system is designed to provide clean connection between components while maintaining modularity and ease of maintenance. To achieve this, the wiring is divided like the system itself, into four quarters. Each quarter of the robot requires two connections to be made: one for the high voltage bus, and one for the low voltage and signal.

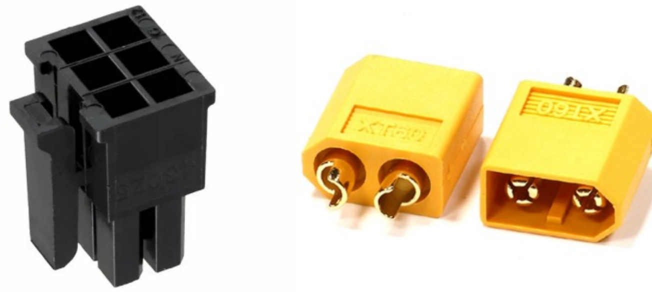


Figure 27: Molex Microfit and the XT60 Connector

The high power connection is made with XT60 connectors in each corner of the robot. These connectors are rated for high current and provide a low impedance connection for the main drivers of the leg. The low power connection is provided from a breakout at the microcontroller. This breakout produces looms of six wires that run directly to each of the quarters. These looms are connected to each quarter using Molex Micro-fit connectors, a locking connector that provides a secure connection even with some amount of force applied by inertia in the robot's movement.

Figure 28: TE Connectivity Connector Assembly

For connecting the signal cables between the Arduino and the breakout board, we chose panel mount punchdown connectors, pictured above. Using the stripless connectors on this assembly, wire harnesses were made in groups of four between the two boards. The goal is to make the process of replacing the microcontroller or servicing the electrical boards as easy as possible; instead of having to reroute every connection, simply placing four connectors into the Arduino completely integrates the system. One concern is that the connectors do not have a matching locking mechanism on the Arduino, it's possible that during operation or maintenance the connectors could come

loose.

5.6. Camera Correction Software

A problem that was encountered previously was the expectation that the camera sensor would tilt when the quadruped is moving. The movement of the camera sensor can likely cause issues with any mapping or vision algorithms employed on the robot. While the rotation may not be noticeable at slow speeds, at higher speeds it may rotate the images enough to the point where certain features are not discernable between image frames, which would result in those image scans leading to bad localization. In order to combat this, a ROS node was written to transform the images based off of the change in angle. An IMU was installed on the robot to detect the current angle of the robot. Because the only thing of interest is the current angle and not a continuous total change in angle, the issue of drift did not need to be taken into account. In addition, the IMU used pairs a magnetometer, gyroscope and accelerometer with all data automatically merged on an off-board chip for very accurate results.

With the IMU data, the images can then be translated appropriately to account for the rotation. If the rotation from one frame to the next is 5 degrees, the image is then rotated -5 degrees to line up with the previous image. The image is also cropped 5% on both the top and bottom, as rotating a skewed image will result in image loss along the horizontal borders. By doing so, 10% of the image is lost, but ultimately the loss in image is worth the performance benefits that it offers.

Code Implementation

Implementing the above image correction in code required interaction between various hardware and software components. The image correction process begins when an image is captured through the Asus Xtion Pro sensor that is connected to the RaspberryPi. When an image is captured, the RaspberryPi checks the data coming in through the serial port. The accelerometer that was mentioned above is connected to the Arduino Mega, and is transmitting the tilt of the robot every tenth of a second over serial. The message being sent has a specific header which is “IMU Data” followed by the angle. The RaspberryPi parses this information and stores the current angle of the robot. It then compares the current angle to the previously stored angle of the robot. The difference between the angles becomes the amount to shift the picture in degrees. The image is then cropped 5% from the top and bottom using built in image functions to

account for the changes that may occur when the image is rotated.

This implementation produces a stream of images that are all translated to maintain a general baseline despite the system undergoing various changes in height and angle.

5.7. Raspberry Pi 3 Setup

In order to get started with the Raspberry PI, an operating system must be installed on the device. There are various operating systems that can be installed such as Raspbian, Ubuntu MATE, Android, etc., but it was decided that the two operating systems that would be reviewed were Raspbian and Ubuntu MATE. These two operating systems were selected because they are the best supported. The pros and cons of each will be discussed below.

5.7.1. Raspbian Jessie

The current Raspbian LTS version is Raspbian Jessie. Raspbian Jessie is a very lightweight operating system (OS) that is the officially supported OS for the Raspberry Pi. As such, it has a significant amount of community support. Raspbian Jessie can be downloaded from the official Raspberry Pi website. The operating system is Debian based, meaning that it is Unix-like in operation. Experience using other Linux based OS' will transfer over to Raspbian Jessie.

5.7.2. Ubuntu MATE

Another popular OS to install on the RaspberryPi is Ubuntu MATE (currently in version 16.04). Ubuntu Mate is a stripped down version of the regular Ubuntu distribution that has been heavily configured and modified to run on the Raspberry Pi's ARM processor. The OS has also been configured to interface with the onboard peripherals on the Raspberry Pi such as the Wi-Fi card without any further configuration. (previous versions did not work so well with the Raspberry Pi 2). Ubuntu MATE takes up about 10% more system resources when idle compared to Raspbian Jessie, but offers benefits in other areas such as ROS integration (will be discussed later). It also works almost identically to the regular Ubuntu distribution when it comes to commands and interfaces. Thus, if the end user has experience with Ubuntu, Ubuntu MATE is a convenient and effect choice for Raspberry Pi integration.

5.7.3. ROS Integration

The biggest factor in determining which OS to use is ROS. It was decided early on that ROS would be used to handle various functions related to the robot and also communication between devices. ROS installation is extremely different between Raspbian Jessie and Ubuntu MATE, so careful consideration should be made when deciding how to proceed. The process for ROS installation on each OS will be discussed below:

5.7.3.1. ROS Integration on Raspbian Jessie

Installing ROS on the Raspberry Pi is a very long process that takes several hours to complete. ROS has yet to make prebuilt binaries for Raspbian Jessie. As such, all of the code and packages must be built from source. The ROS website has provided extensive documentation on how to install the base ROS components on Raspbian Jessie. Installation of the base ROS components is very straightforward and only takes about an hour to install. However, the installation only covers the base ROS packages, and does not include many of the packages required for this project. Specifically, the base installation does not come with the Turtlebot packages. The base ROS installation is comprised of about 50 packages. When the Turtlebot packages are downloaded and added, the number of packages goes up to around 600-700. Many of these packages are large programs, such as OpenCV, and require a significant amount of time to build. Specifically, trying to build the ROS base with the Turtlebot packages can take roughly 6 hours. In addition, the build may terminate due to various dependency issues that may occur, or simply because the Raspberry Pi runs out of memory when it tries to build the larger packages. Furthermore, if the build does succeed and then another package is added to ROS, the entire process must be repeated and all of the packages will need to be built again. Because of the significant amount of time required to build the packages and ROS, it is not recommended to use Raspbian Jessie as the OS, as it will result in losing a significant amount of time in waiting for the packages to be built.

5.7.3.2. ROS Installation on Ubuntu MATE

Since the release of ROS Kinetic, several ROS binaries have been compiled for various Ubuntu distributions. This means that ROS Kinetic (previous ROS versions are not supported) can be used on an installation of Ubuntu MATE. Installation of ROS on Ubuntu MATE is the same as how ROS would be installed on a standard computer OS. Instructions on how to install ROS Kinetic can be found on their website. Installing the base ROS packages will take roughly half an hour on a 300KB/s download speed. After

the base packages are installed, the Turtlebot packages must be obtained.

Documentation on installing the Turtlebot packages can be found on their website, but it is important to note that the instructions on their website only show how to install the packages for ROS Indigo. In order to install the ROS Kinetic Turtlebot packages, all of the commands must be changed to reflect the ROS distribution you are using if a ROS distribution is specified. For example, the documentation on the website specifies to use the following line:

```
sudo apt-get install ros-indigo-Turtlebot
```

The ROS Kinetic variant of this line would be:

```
sudo apt-get install ros-kinetic-Turtlebot
```

Once the Turtlebot packages are installed, ROS should be fully configured to work.

5.7.4. Operating System Installation and Configuration

Once the operating system has been chosen, it is time to install it on the Raspberry Pi. In order to install the operating system, the image file of the OS must be written to a microSD card. The operating systems and ROS combined will take about 7GB worth of space on the microSD card, so it is recommended to use a 16GB microSD card for the project, as this will allow for ample storage space to be used for various applications. Furthermore, additional storage space can be used as swap space (discussed below) in order to ensure that the Raspberry Pi does not run out of memory. For first time setup, a display will be needed to see what the RaspberryPi is currently doing. Please note, when the OS is not installed, it is not possible to use many applications such as SSH or VNC to connect to the Pi. Therefore, a display should be used to show the end user what they need to do to install. When the Pi is booted with an OS, there will be an option allowing the user to install from the SD card. After some time, the OS will be installed on the Pi and the user will need to go through the configuration steps for the OS they are using. Once the OS is installed, many packages will need to be updated and installed. It is important to pay attention to the documentation provided by ROS when it comes to installing packages, as many of the packages are not available when the OS is installed. Failure to install the packages will result in various packages not working.

5.7.5. Swap Space

In order to maintain efficiency and prevent any crashes, it is important to enable swap space on the Raspberry Pi. The Raspberry Pi 3 comes with 1GB of RAM, which is a significant amount of RAM for most functionality. However, because this project requires the use of computationally heavy applications such as GMapping and SLAM, it is important to allocate swap space. Swap space is storage space that the device uses as virtual memory. As the device nears its RAM limit, it stores unused files in the swap space, effectively releasing some RAM and preventing the device from crashing due to too much consumption. On the RaspberryPi, the swap space can be allocated from unused space on the SD card. It is important to note that constantly writing to the SD card can burn it out (they have a limited number of read/write cycles), so caution should be given when creating swap space. For the purposes of this project, only 100MB of swap space were added. This resulted in less crashes and memory errors when building large files or running mapping algorithms. Therefore, it is recommended to use swap space on the device. A good resource for allocating swap space can be found here.

5.8. GMapping

After various tests with different mapping algorithms, it was determined that the ROS provided GMapping package will work best with the robot. However, the GMapping package takes up a significant amount of system resources. On the Raspberry Pi 3, RAM usage increased to 95% while mapping a room and crashed occasionally when the system ran out of memory after about 13 minutes. In order to prevent this, various changes to the GMapping package must be made to the specified GMapping attributes below:

throttle_scans (5)

This attribute tells GMapping to process 1 out of this many scans

angularUpdate(1.5)
linearUpdate(2)

These attributes specify when to process new scans. The angularUpdate attribute processes a new scan after the robot's angle changes this many degrees, while the linearUpdate attribute processes a new scan when the robot moves this many inches.

map_update_interval(7)

This attribute specifies after how many seconds to update the map in applications such as RVIZ. Setting this to a higher number results in better performance as the node is not transmitting all the data it obtained, but results in RVIZ updating even slower than usual.

Changing these values, as well as enabling swap space, resulted in a total of 70% RAM usage, which leaves enough RAM to run almost any other application needed and prevents the Raspberry Pi from crashing. Yet, by changing these values, a loss in accuracy might occur. Initial tests with the above values resulted in no noticeable accuracy loss. If the accuracy loss is significant, it is best to modify the angularUpdate and linearUpdate attributes, as those will affect how the odometry interacts with the laser scans.

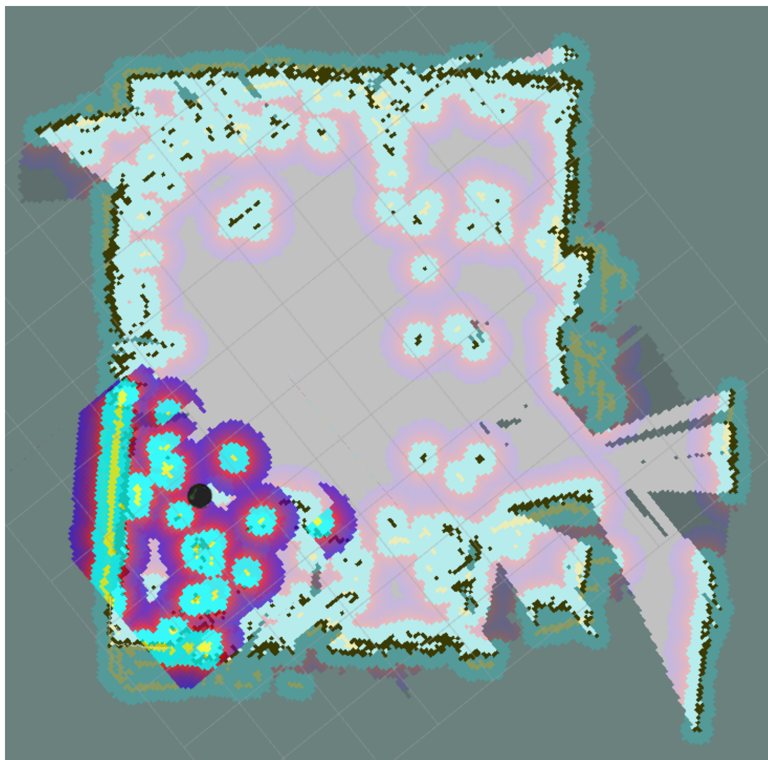


Figure 29: GMapping Results from Outer RBE Lab Room

The above image shows the map generated from driving a Turtlebot around the outer room in the RBE lab with the aforementioned settings. The mapping of the room is fairly

accurate and does not have any glaring, immediate issues. An important factor in the accuracy of the room is the odometry data. Although GMapping can work without odometry data, it is encouraged to use odometry data. When GMapping is run without odometry, it only uses LaserScans to generate the map. Many times, the scans do not match up, which results in the robot not being able to localize itself. In addition, if sharp turns are made, the LaserScans will fail due to the fact that the rotation was too large for the LaserScanner to keep up with. If this happens, the robot will believe it turned a greater/lesser amount than it actually did. Therefore, some form of odometry data is needed. The quadruped that is being developed is not a wheeled robot, so an alternative way of tracking how far the robot has walked needs to be implemented. Since the robot provides feedback on where all of its joints are (we are using two servos with ranges 0 to 180, effectively), calculations can be made to determine how far the limbs have moved. This data can then be transmitted through ROS to the GMapping node to use as supplemental information. Even if the data is not precise, it will still be very useful to the GMapping algorithm, as the GMapping algorithm uses the odometry data to calculate the error from the LaserScans.

GMapping on MUTT

As mentioned above, the GMapping package works best when odometry data is supplied. In order to test the image correction and mapping capabilities of the robot, an experiment was designed to map a small enclosure using the MUTT robot with odometry data from the Turtlebot. The purpose of this experiment is to validate the previous findings that the robot would be able to map a room despite the changes in position and angle that it undergoes when walking around.

The design of our experiment was as follows. One tester hovered over the Turtlebot with the MUTT robot in their hands. The Turtlebot base was connected directly to the RaspberryPi and driven from a base computer by another tester. The odometry data produced by the Turtlebot was broadcasted to the GMapping package in the standard way. As the Turtlebot moved forward, one of the testers rocked the MUTT robot from side to side and also moved forwards, inducing motions similar to what the robot would undergo if it was in motion. When the turtlebot turned, the tester also rotated with the MUTT robot and continued rocking it back and forth. The result of this experiment was a mapping of the small enclosure.

5.9. VisualSLAM

VisualSLAM is another method of performing SLAM on a robot. VisualSLAM is different from GMapping in that it relies more on hard feature extraction instead of filters and particle weights like GMapping. VisualSLAM offers superior performance in unstable environments or in situations where the frames between samplings change. This superior performance is attributed to the fact that VisualSLAM detects more prominent landmarks/features instead of randomly sampling like GMapping does. Because of this, it is a viable mapping algorithm for the MUTT robot, as the camera frames can be very inconsistent.

While there are many VisualSLAM algorithms out there, RGB-D SLAM is among the most popular, well documented, and supported by ROS. Therefore, RGB-D SLAM will be tested on the MUTT platform.

5.10. BreezySLAM

BreezySLAM is an open source, beginner level, GUI oriented approach to SLAM. It was

designed as an introduction to SLAM and contains modular features that allow it to adapt to various mapping situations easily. The algorithm was created with microcontrollers in mind and is thus very lightweight and can be implemented in four different programming languages. The algorithm works like GMapping but handles updating and particle weights differently in that it does not readily delete particles that have low probabilities/weights. With this implementation, many changes can be made to the particle filters and their characteristics, and the package even allows for new filters to be created. Using the standard BreezySLAM implementation, the following mapping of the same enclosure using the same experiment was created.

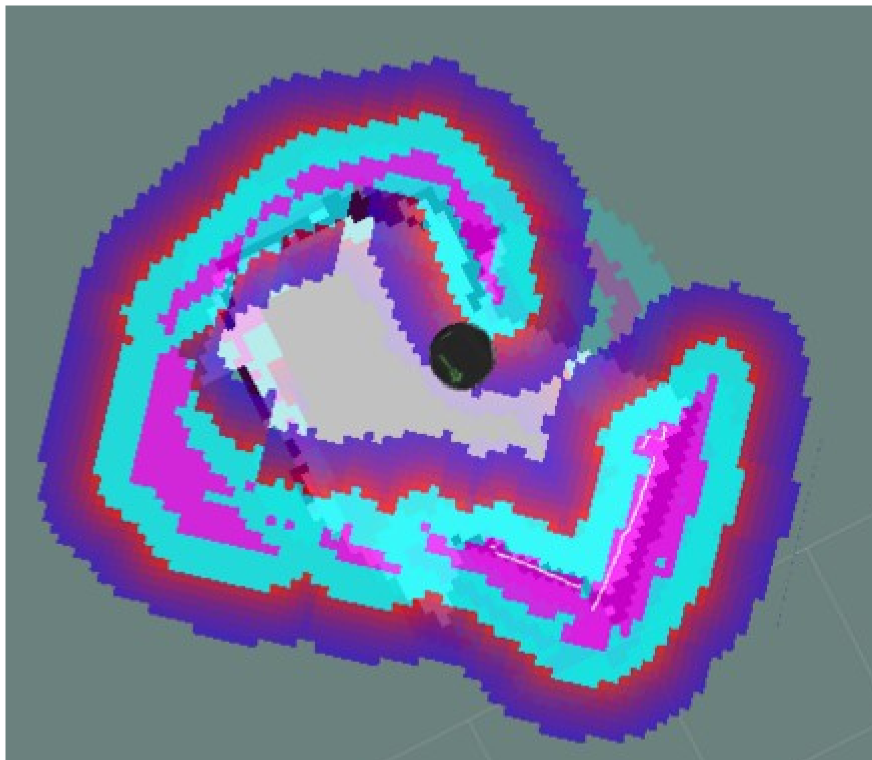


Figure 31: Map generated by BreezySLAM

5.11. HectorSLAM

Before the design was switched to using two servos, it was not believed that proper odometry data would be provided from the robot. Therefore, SLAM alternatives that did not rely on odometry data were researched in order to find a SLAM solution that could work with our design. Of the many SLAM algorithms that work without odometry

data, HectorSLAM produced the best results. The image below shows the results from mapping the first floor hallway of Atwater Kent:

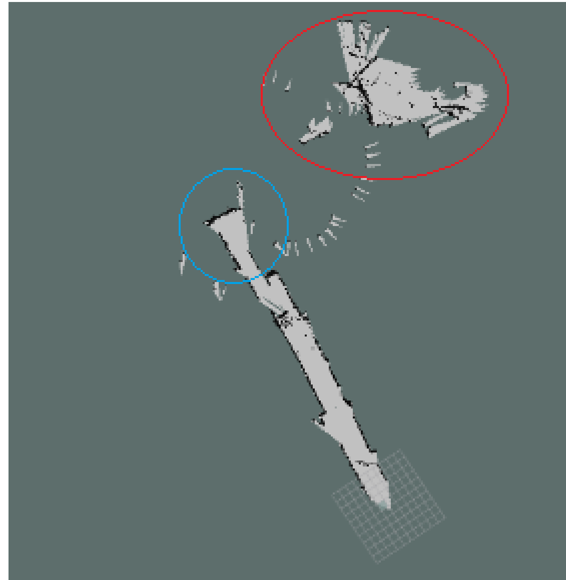


Figure 32: Mapping from HectorSLAM

We can see that the hallway itself was mapped properly when running HectorSLAM. However, issues were encountered when attempting to turn corridors. HectorSLAM relies solely on the LaserScans, so when the robot rotates at a fast degree, the LaserScans do not match up as it cannot recognize the same features as in the previous frames due to the rapid change in angle. Because of this, the robot loses its position and thinks it rotated or translated into another area. We can see this happening in the image above in the image above. The end of the hallway is indicated by the blue circle. At this point, the robot can either turn right and enter the room to the right or turn left and continue down another hallway. In this case, the robot was turned to the right. When it turned, it lost track of its positions and the result of this error is what is shown in red. Due to the rotation, HectorSLAM thought it had traveled a significant amount and generated an incorrect mapping of its environment. Many attempts were made to try and mitigate this error, such as changing the frequency of sampling, changing the maximum rotation angle allowed, and simply lowering the angular velocity of the robot. Despite these changes, rapid changes in angle still continued to affect the mapping adversely.

Despite the issue with rotating, HectorSLAM should still be considered a viable alternative to GMapping. The mapping algorithm was tested using an ASUS Xtion Pro, which is an RGB-D camera. As stated by the documentation, HectorSLAM works best when it is run with LIDAR. When run with LIDAR, large changes in angle do not seem to cause as many problems. In addition, an IMU can be paired with the HectorSLAM algorithm as supplemental information. The IMU would transmit the current angle of the robot. If the current angle changes quickly, it can inform the SLAM algorithm and the algorithm will use this information to map the room based off of where it thinks the LaserScans match up given the current angle. Further work in this area should consider using a low-cost LIDAR sensor for mapping.

5.12. Voice Commands

The control schema behind many robots revolves around point and click controls on a GUI interface like RVIZ. While this method is effective and gives precise control and visualization of what the robot will do, it is lacking in terms of personal control. Having to open up a GUI program like RVIZ, enable screens and topics, and point and click where the robot should go is counterintuitive and goes against the mental model that many people have regarding robots. Many people perceive robots as intelligent machines that can be voice controlled and require little human in the loop control. This is the path that many robots are taking, and in order to make the system modular and easy to use as a research platform, it was decided that voice control would be implemented. Voice control was implemented in the following way:

Speech Parsing

The main difficulty with implementing voice commands revolves around parsing the speech input. In order to parse speech, the microcontroller/computer needs to be constantly listening and storing incoming speech onto the disk in the form of some audio file. This audio file then has to be decoded and deconstructed, and every word or series of words needs to be parsed and checked against valid commands. While this is largely a solved problem, it requires time and disk space which in a microcontroller can be quite costly. There are many libraries and API's that automatically do the parsing either on the device or by off-loading it to a cloud service, such as Google Speech API or Amazon Web Services (AWS). There is a tradeoff that must be considered; either the computation is done on the device which takes up disk space and processing time, or the parsing is done on a cloud service, which requires a constant internet connection

and is asynchronous. Each method has its merits, but ultimately, it was decided that in order to make the robot more modular and have extended applications, all of the parsing would be sent through Amazon's artificial intelligence, Alexa.

Alexa Integration

Alexa is an artificial intelligence that runs on the Amazon Echo products. Alexa is capable of performing various tasks that are beneficial to the user such as setting alarms, ordering food, checking the weather, playing music, etc. The artificial intelligence is controlled through voice commands and an app. Amazon released the Alexa AI to the public and allows developers to create "skills" (functions) for the AI through their developer network. The Alexa app works by constantly listening for a wake word that triggers the device to turn on. In this case, the wake word is "Alexa". Once Alexa is triggered, she begins to record all communication after the wake word for a short period of time. This recording is then sent to AWS for parsing, and the parsed input is then sent back to the device to be used in whatever manner desired. At this point, we can integrate the functionality that is needed to control the robot.

Integrating Alexa with the Raspberry Pi

Amazon released source code with full instructions on how to install Alexa on the Raspberry Pi. This information was released to allow developers to implement Alexa on a host of devices, however, the documentation does not cover exactly how to integrate this functionality with low level control through the Raspberry Pi. In this section, we will not discuss how to install Alexa as there is already strong documentation available for said procedure on the internet. Rather, we will focus on how to obtain input on the Raspberry Pi from Alexa.

Setting up the Alexa Skill

In order to control the robot, we first need to know what kinds of commands will be issued. By signing into the Amazon Developer Network and navigating to the Alexa portal, new skills can be added. Every skill has a custom trigger phrase that tells Alexa to interpret voice input relative to that skill. In the case of the MUTT robot, the trigger phrase is "Alexa, ask MUTT to…" where the following voice input is the command to run. Any trigger phrase can be used as long as it follows the restrictions set by Amazon. Every skill has a custom "interaction model" which is a specification of the allowable

inputs and functions that the Alexa AI can pick up on and interpret. An example of the MUTT robot's interaction can be seen below:

```
    "intent": "GetTurn"
  },
  {
    "slots": [
      {
        "name": "mode",
        "type": "LIST_OF_MODES"
      }
    ],
    "intent": "GetMode"
  }
```

Figure 33: Interaction Model

In this model, the intent (function) is "GetMode", which has a variable called "mode" that is made up of a type called "LIST_OF_MODES". The LIST_OF_MODES is a list of strings that represent the modes that the robot has such as start, stop, initialize, walk, etc. The interaction model can be seen as a template for what the user can ask the Alexa AI.

Utterances

Utterances are the sentences or words that users say in order to invoke functionality through the skill. Some sample utterances for the MUTT robot are "Alexa, ask MUTT to initialize" or "Alexa, ask MUTT to walk forward 10 feet". Amazon makes it easy to make generic utterances by allowing the developer to map variables and functions to expected input. A list of utterances available for the MUTT can be seen below:

Sample Utterances

These are what people say to interact with your skill. Type or paste in all the ways that people can invoke the intents. Learn more

Up to 3 of these will be used as Example Phrases, which are hints to users.

```
1  GetDirection move {Direction} {distance} {Unit}
2  GetDirection advance {Direction} {distance} {Unit}
3  GetDirection walk {Direction} {distance} {Unit}
4  GetDirection travel {Direction} {distance} {Unit}
5  GetTurn turn {Dir} {distanceAngle} degrees
6  GetTurn turn {Dir} {distanceAngle}
7  GetTurn rotate {Dir} {distanceAngle}
8  GetTurn rotate {Dir} {distanceAngle} degrees
9  GetMode {mode}
10
11
```

Figure 34: List of Utterances

As portrayed by the figure above, many utterances can be and should be written to maximize the coverage of all of the possible input from the user.

Communicating with AWS and Alexa

With the basics covered on what goes into a skill, we can now talk about how to communicate with AWS and Alexa. Every Alexa skill requires a host address, meaning it needs a place to send its requests to and from. Most publicly available Alexa skills have a custom domain that requests are sent to, but this requires a capital investment to host and comes with a significant amount of overhead. In order to bypass this, we will be hosting our skill through our own internet connection via localhost. We can do this by exposing a port that the skill listens for. This can be accomplished in many ways, but this document will focus on using ngrok due to its open source and free nature. Ngrok is a tool that exposes tunnels to localhost to be used by the outside world. Running the ngrok took provides the user with a URL that can be accessed from anywhere. This is needed as the Amazon skill requires an endpoint to be set when the skill is being setup, as this is where all of the data parsed will be sent to. Once the tunnel has been exposed, there will be a connection between AWS and the user that Alexa can communicate through.

Setting up the Communication Server

Once the tunnel is open, a web server needs to be run on localhost. This server is responsible for sending POST requests to and from the Alexa device and AWS and parsing the returned input from AWS. This web server can be implemented using a python library called Flask-Ask. The Flask-Ask library facilitates communication

between the Alexa device and AWS by managing sessions between the two components and allowing the user to send direct intents and queries through HTTP requests. It also allows for the data that is being sent back from AWS to be parsed easily as all the data is stored within the session and can be called at any time. When the right data is obtained by the session, it can then transmit the proper serial commands to the Arduino in order to control the robot's motion. For example, once the session obtains the "initialize" phrase, the RaspberryPi can send a serial message to the Arduino in order to start the robot up.

5.13. Navigation

A navigation stack was written for the robot to aid in navigation and mapping. The navigation stack was written to work in the ROS environment. It takes incoming data from the map being generated by GMapping, and generates a grid of nodes based off of the height and width of the map. Each grid cell represents a node and has a certain weight attached to it that indicates whether that node is an obstacle, open space, or unknown. A global list of nodes is generated and all nodes that are obstacles are removed from the list in order to ensure that those nodes do not obstruct with the path planning algorithms. Once the list of nodes is generated, any nodes that fall on or near the boundary between known and unknown areas are added to a list of frontiers, which are areas that have yet to be explored by the robot. These areas are important to explore as they allow the robot to find the locations it needs to path to in order to fully complete the map of its environment. Once a frontier is found, the robot searches for a path to get to that frontier. The search algorithm used in this case is A*, as A* allows for the fastest path to be found in a grid. Other search algorithms can be used such as Dijkstra's Algorithm, but since A* uses an admissible heuristic it was believed that it would produce the best results.

Once the path has been generated, a series of waypoints are extracted from the path that the robot must navigate to before reaching the goal, and then from that waypoint, the path to the end goal is computed again. These waypoints are typically areas in which the robot must turn to get to the next goal. The reason for doing this is that by navigating to the intermediate waypoints instead the end goal, the amount of errors are reduced. For example, when navigating to to the end, if an obstacle appears in the environment that obstructs a portion of the path, then the robot will not be able to continue on till the end goal since the path was never calculated. By navigating to the

waypoints, once the robot reaches its first waypoint, the path to the end goal will be recalculated and would create a new path that goes around the new obstruction. This approach ensures that even with a dynamic map, the robot will still eventually be able to navigate towards its end goal.

Navigation with Alexa

Alexa was integrated with the navigation stack in order to allow the robot's position to be controlled. The user can give the robot a distance and direction to travel to, and also an angle to turn to. The navigation code takes the current map data and determines the width and height of each cell. With this data, the distance to travel by is then divided down by the dimensions of the grid cell. Using trigonometric functions, the amount of nodes to travel in the X and Y dimensions is obtained based on the robot's current position and location. With this information, the correct node can then be pulled from the list of nodes that is closest to the distance that was input by the user. If the node selected is an obstacle, the user is informed and asked to give another distance.

Patrol Mode

As a way to expand upon the functionality of the robot, a patrol algorithm was generated. The patrol algorithm works by interfacing with RVIZ. The user selects a series of points to navigate using a special topic (pGoal). The user can select an arbitrary amount of points to patrol. The navigation stack takes these nodes and tries to generate a path between each node selected. If no path could be generated, an error message is displayed. If a path was found, the robot will begin to navigate to the end goal. Once the robot reaches its end goal, it will update the weights of the nodes in the path it took to random values. By doing this, the A* algorithm will try to take an alternative path since A* takes the weights of each node into account when generating a path. The result is a different path to the end goal every time the robot completes one cycle of the entire path. This navigation mode ensures that an area can be patrolled in a dynamic way.

5.14. Facial Recognition & Following

As a way to demonstrate the computer vision capabilities of the robot, facial recognition code was written. Using OpenCV and the ASUS Xtion Pro camera, a facial recognition node was created. OpenCV is a collection of open source libraries for computer vision. One of the libraries it provides is the HaarCascades library. The

HaarCascades library is a series of functions and filters that can be applied to images in order to detect various features. One filter that it provides extracts all detected faces from an image. The filter was created by running machine learning on hundreds of images. From these images, the creators constructed several filters that determine whether or not something is a face. Because the filter was created through machine learning, it is able to determine if something is a face from different angles which means that the people in the image do not to be directly looking at the image.

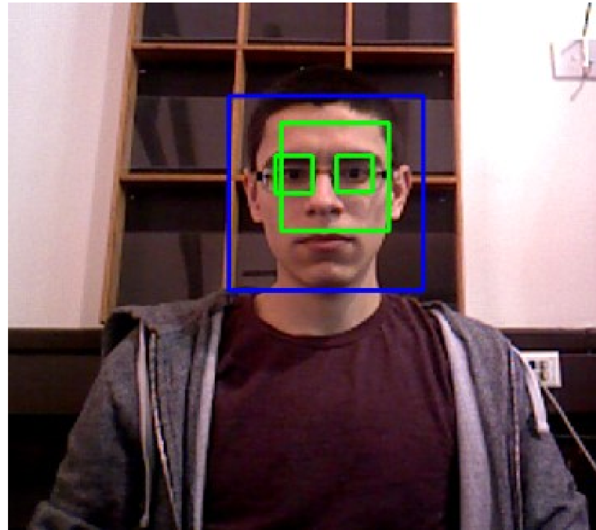


Figure 35: Facial Recognition Output

These filters are applied to the image data streams coming from the ASUS Xtion pro, and the bounding boxes of the detected faces are rendered to the screen. In addition, we apply filters to the eyes in order to determine if the user is looking directly at the camera. The end result is having a node that processes the data coming from the image sensor and determining whether or not there are people that are looking at the robot. The node could be used for various Human-Robot applications such as following a human face, determining mood, or determining if the user is paying attention to the robot.

5.14.1. Following

The intended purpose of the facial recognition node was to use it to follow a human face. Many studies show that dogs recognize faces and can differentiate between known faces and unknown faces. In addition, dogs recognize humans through sight and smell. Since the robot is designed and modeled to mimic a dog, it was

determined that facial recognition would be used to create a program that would move the robot towards the human that it detected. In order to do this, the robot needs to know how far to travel and when to stop. The distance to travel can be determined by analyzing the image frames. At time t, the bounding box of the detected face will be at a certain position. At time t + 1, the bounding box can either be further away (smaller), closer (bigger), remain the same, or be gone. If the bounding box of the detected face is becoming larger, this means that the user is moving towards the robot. If the bounding box of the detected face is moving away and becoming smaller, then that means the user is moving away from the robot. At this point, we then trigger the robot to start moving in the direction of the face. The rate at which the robot moves can be calculated from the images. If the sampling time of two consecutive frames is known and the change in position in pixels is known, then using simple algebra, the general speed of which the person is moving can be estimated. The robot is then passed the value and moves at that speed until a certain threshold is reached. The threshold in this case is whether or not the face is getting smaller. If we are moving at the same rate or faster than the end user, then at a certain point, the bounding box of the detected face will either stay constant or reach a certain dimension that can only be achieved when the end user is a close distance away. When these conditions are reached, the robot either stops or slows down.

While the above method worked during initial tests, there are a lot of problems with the system. The camera that is being used can only accurately detect faces up to 6 feet away. In addition, overhead lights seem to cause issues with the depth maps that are used to determine the distance/dimensions. However, the biggest problem comes from the static camera sensor. On the Turtlebot and the quadruped, the camera sensor will be mounted statically. This means that it cannot pan or tilt in any direction. Therefore, if the detected face goes out of frame in any direction, there is not much that can be done besides rotate in place or move backwards/forwards in order to try and locate the face again. This problem can easily be solved by installing a pan and tilt system on the robot and mounting the camera on top. This would provide the robot with a “neck” that would allow it to rotate the camera sensor much like humans do when looking around.

6. Results

6.1. Actuators

6.1.1. Motors and Controllers

During the initial rounds of testing, problems were encountered using the Pololu motor controllers. The motor controllers were found to consistently enter their shutdown state due to overcurrent protection mechanisms. The controllers failed to provide adequate current to drive the Vex 393 motors. Testing to find a suitable replacement was done with on hand materials. The Vex 29 motor controllers were chosen as a replacement due to their availability. Results of initial testing showed current handling in excess of 3A for time greater than 5 seconds.

In practice, however, the Vex 29s did not stand up to the demands of the Mutt robot. This is likely attributable to three characteristics of the system: operating voltage, current requirement, and duty cycle. To start, the Vex 29's are technically rated to operate at 8.5v, the typical voltage for most vex equipment. While testing showed that the controller were able to operate at 12.5v (the voltage of our system), it seems that extended use forces the devices into a similar shutdown state as the Pololu controllers, with symptoms of reduced output and total shutdown. This was likely exacerbated by the excessive current drawn by our motors and the duty cycle of their operation; at times resting the vex motors for several seconds at stall. The total power going through the controllers being in excess of 40 Watts, thermal shutdown is the conclusion we drew from the symptoms of the controllers.

The effect of the failing motor controllers was that the upper limbs remain inactive at this part of the project. It was decided to exclude their use entirely, to ensure that the system as a whole was as reliable as was feasible.

6.1.2. Servos

The lower joint of each limb was actuated by a series elastic tendon controlled by a high-torque servo motor mounted on top of the robot. The servos were typically strong enough to drive the desired position of the lower joint. Initially it was found that the servos reached their maximum torque while driving the lower limb. In order to better actuate the joints, the cams attached to the servos were reduced in radius to apply a

greater force for the given torque of the motors. After reducing the radius of the cams, the force was enough to properly actuate the joints for most cases.

In testing gaits, the wire rope used for the tendons held together under the applied loads. Because guides were made to direct the cabling, minimal scoring of the plastic occurred. Issues did arise, however, as the force on the cable sometimes caused the shaft collars used to slip. This led to an increase in cable length and, thus, a decrease in available pulling tension to the lower leg. The collars were simply re-situated on the cable and tightened again during testing. After the final desired length of the cable was determined, the collars were replaced by copper swages just as were installed at the cams on the other end of the cable.

In the case that the robot rests primarily on one of the legs, the servos were not strong enough to realign the robot to level. Instead, the gaits were adjusted to reduce the weight on any one leg.

6.2. Microcontrollers

6.2.1. Arduino Due

Problems were encountered with using the Arduino Due for I2C communications with the encoders. The encoders that were used were daisy-chained together. The initial encoder was tied to the Arduino Due SDL and SCLK lines, which were the primary lines for communication. We found that the signal coming from the SDL lines on the Arduino Due was not sufficient to address the series of encoders. The first encoder in the chain would initialize properly, but the rest remained in an indeterminate state due to the fact that they were not receiving an initialization address.

This issue could have been caused by many reasons, but we believe that the Arduino Due was unable to provide the necessary voltage and current to communicate with four encoders. The Arduino Due, unlike many of the other Arduino boards which operate at 5V, operates at 3.3V. As a result of the lower voltage, the board may not have been outputting the proper signal. In order to verify that the board was indeed the problem, we tested the system on an Arduino Mega. We found that the Arduino Mega was able to properly address and initialize all of the encoders using the exact setup that was being used before. Because of the aforementioned issues, it was decided that the Arduino

Due would be switched for an Arduino Mega. There were some concerns with making this switch such as memory space and overall system performance that will be discussed below.

6.2.2. Arduino Mega

Overall, the performance of the Arduino Mega was suitable for the robot's needs. This was evaluated and confirmed as follows.

Clock Speed

There were initial concerns that the Arduino Mega would not be able to execute fast enough to run in real-time. The Arduino Mega is rated to run at 16 megahertz, while the Due operates at 84 megahertz, so there was a considerable downgrade in clock speed. However, during initial rounds of testing, the Mega was able to execute all of the system needs in time. The main concern in using the Mega was if it would be able to manage the calculations required to operate the robot in less time than the required interrupt period.

The team was surprised and impressed with the performance of the Arduino Mega. Operating with a 10ms interrupt rate, the device was more than capable of handling the tasks required for smooth operation of the robot. Testing showed, in fact, that the primary delay in operation of the code was communication with the I2C encoders on each quarter. It was found that addressing and communicating with each encoder took 1.6 milliseconds, so in total, communicating with the encoders took up 6.4 milliseconds, which left the Mega with enough to execute the other portions of the software.

Electrical Characteristics

The Arduino Mega operates at 5 volts natively, meaning that it could interface directly with the DC-DC converter specified for the Mutt. The nature of the other logic level components involved also meant that the Arduino would have to interface with these components at 5v. It was found with the Arduino Due - whose operational voltage was 3.3v - that communication with the other components would not behave correctly. The Arduino Mega, however, functioned exceptionally well with the other components, both over I2C and PWM.

Architecture and Trajectories

The Arduino Mega has 8KB of SRAM and 256KB of Flash memory. The Arduino architecture is such that data stored before system boot is stored in the Flash memory, and any data allocated during operation is stored in SRAM. All memory management desired during execution of the code must be handled manually by the user. For basic operation of the Arduino, the 8KB of SRAM is plenty to work with; longer trajectories, however, measure in the KB individually and loading an entire trajectory into memory would crash the processor. Our use of Flash memory to store and read only the portions of the trajectory needed immediately successfully managed to store large data types.

6.3. Electrical

The circuits built functioned as expected, after a short debugging period, the primary focus being the breakouts for each quarter, and the main distribution board on the bottom panel.

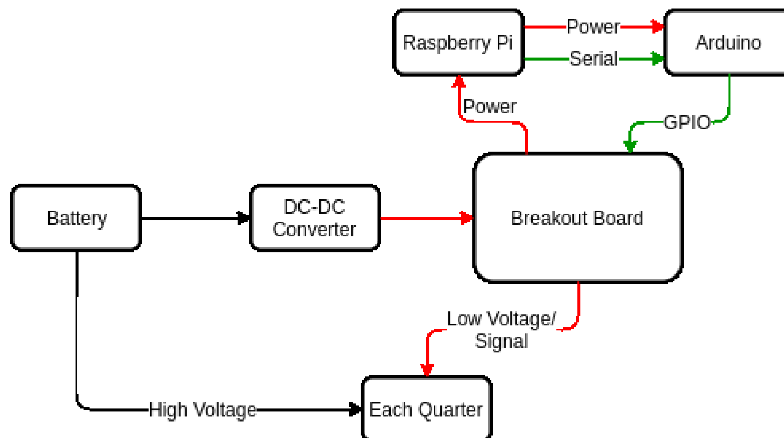


Figure 36: Electrical Diagram

The design showed to be especially useful during the debugging period due to the frequency with which the device was disassembled and reassembled. The ease with which we were able to do this can be attributed directly to the use of multi-conductor connectors between the quarters and the main board. Because only four, secure connections had to be broken to open the device, little work was required to maintenance it. The connectors were assembled by hand, which would typically be an undaunting task; this team, however, did not have access to the tools required to properly terminate the ends of the wires entering the connectors, and had to do so manually. Properly crimping pins onto the rather small wires was not without challenge,

but in the long run, the connections were secure and sturdy.

Also installed as part of the circuit was capacitance inline with the DC voltage supply both at the quarters and at the main board. The primary purpose of this capacitance was to protect the logic circuitry from the electrical noise generated by the motors and servos. During initial testing it was theorized that backEMF from the noisier components caused damage to control circuitry. After adding the capacitance and changing control boards to a more robust Arduino Mega 2560 zero components were damaged or destroyed.

One thing worth noting is the success of the punchdown connectors for interfacing the Arduino with the main breakout board. The modular connector system made it easy to replace boards during iteration changes, or to service the main breakout during debugging. Initial concerns on the security of the connections were alleviated after several weeks of testing without issue.

6.4. Motion and Motion Planning

The motion planning done in Matlab was able to effectively produce motion in the robot. The ease with which they could be produced, however, varied with the complexity of the motion desired. While the basic framework of the motion planner made it trivial to push incremental updates to the robot, actually building motions in the planner was challenging due to the nature of the goal setting. This version of the planner requires the user to manually set the joint positions of each of the limbs, a challenging task for any but the most experienced user.

Gait experimentation on the Mutt platform was hindered by the performance of the primary actuators on the upper limb. The inability of the motors to support the inertia of the robot meant that the actuation was limited to the lower portion of each of the limb, driven by tendon-elastic connected to the servos on top of the robot. Furthermore, it was found that while the servos driving the lower limb were strong enough to lift the robot when working together, when the robot knelt to one leg in the middle of a step, that joint was not strong enough to realign the robot. This meant that the gaits had to be designed around the ability of these joints to actuate the weight of the robot as well as a static upper joint.

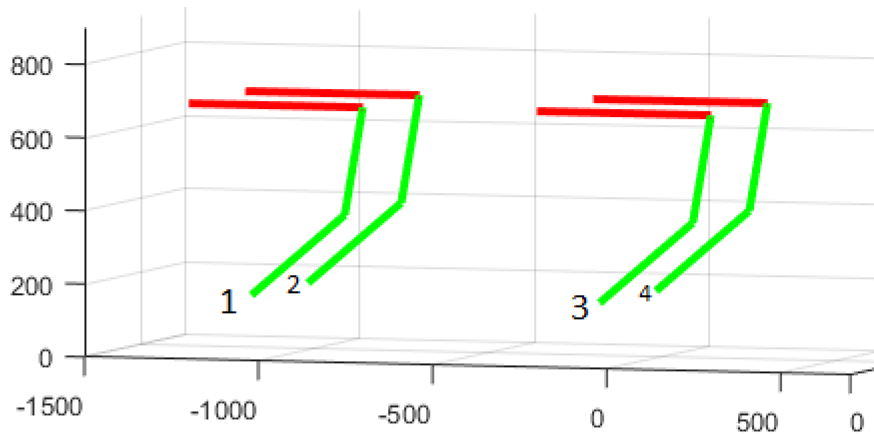


Figure 37: Gait planning output, with numbered legs

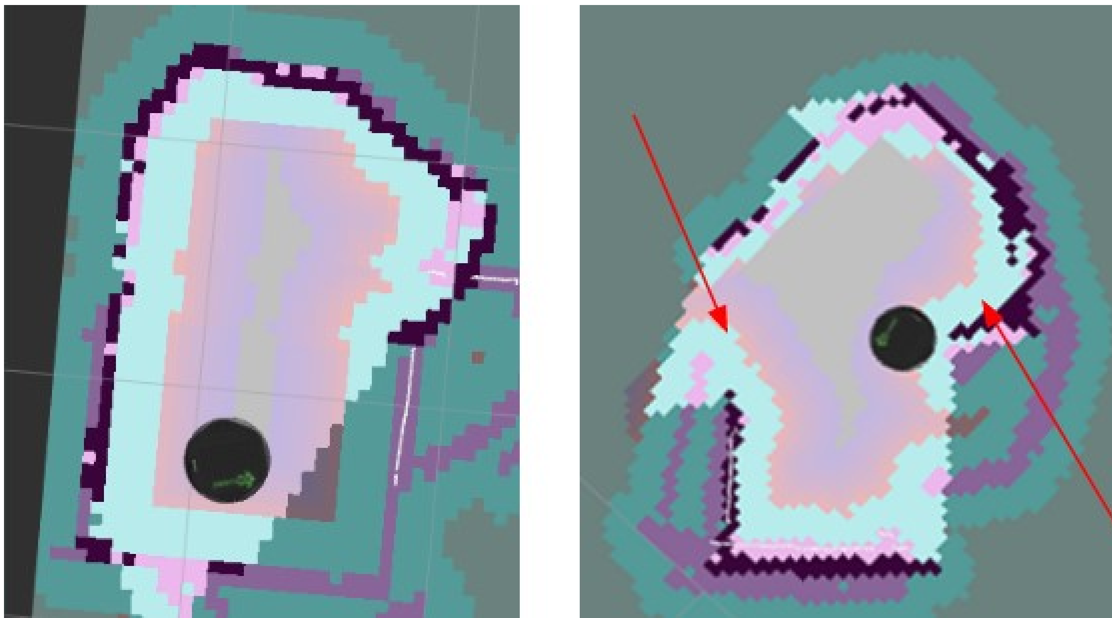
After some experimentation the robot was able to take a step. Further optimization of the gait was able to produce a steady forward motion of the robot using only the lower portion of each of the limbs. Two different gaits were tested in development, differentiable by the order in which the limbs are moved forwards. The first motions attempted were based on the trotting motion of a dog, wherein the normal forces on the robot's legs are centered on limbs 1 and 4, as labeled in the diagram above. The second were based on the amble, where the robot would lift one side (i.e. legs 1 and 3), then move the front leg forwards while pushing with the rear leg.

The diagonally aligned forces of the trot were meant to afford the limbs the ability to swing freely in the air. It was found, however, that the robot was not strong enough to realign the torso after tilting towards one of the legs. This meant that the trot gait produced little forward motion, the average displacement being forced by the grip of the foot pads.

6.5. Mapping

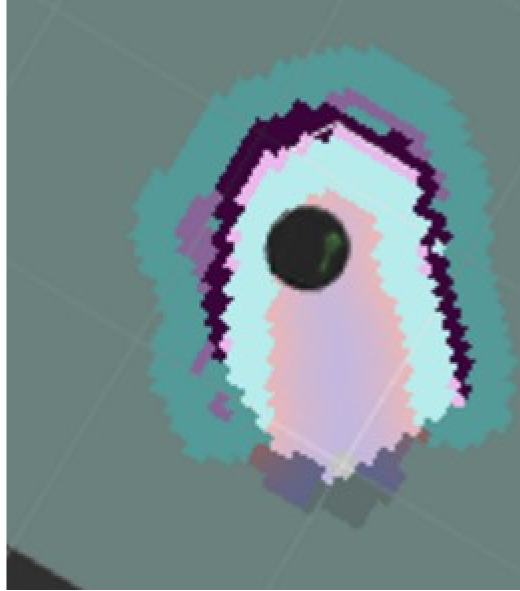
6.5.1. GMapping

Overall, the maps produced by GMapping were the best results that were obtained. The image below shows the initial results from mapping a small environment using our method and the mapping obtained from a Turtlebot:



From the image above, we can see the accuracy of the mapping when the robot is moving. Overall, the robot was capable of mapping the same general outline of the enclosure it was in, but there were two instances where the mapping clearly went wrong, as indicated by the red arrows in the images. At these two points, the robot was trying to turn in place, which introduces the most amount of error into the mapping. This is because the robot’s camera is moving back and forth quickly while rotating slowly, which results in the robot believing that it not only rotated but also translated in some direction. There are also instances where the scans do not match up between frames and so GMapping ignores the incoming laser data and relies solely on odometry, which also can introduce error. This is the main problem with GMapping and many of the other SLAM algorithms because of how much the frames are changing, even with the image correction in place.

After obtaining these initial results, various other changes were made to the GMapping configuration to try and obtain better results. These changes included modifications to the angular rotation values, number of particles, and the minimum score needed in order to consider a scan as being “good” so that less scans are thrown out due to rotation of the camera. This resulted in slightly better results which can be seen below:



The results from this second test were much better than the first, but there are still issues with turning. As seen in the image, there was a slight error in how the top portion of the enclosement was mapped. However, the results produced were much better than previous attempts and show that GMapping can be successfully done on the MUTT platform.

6.5.2. HectorSLAM

HectorSLAM, while being very useful when no odometry data exists, produced very poor results for our application. The HectorSLAM algorithm was able to properly map the corridor with the image correction software, but once the camera begins to rotate in place, the algorithm gets trapped in a locked state and immediately starts mapping incorrectly as it cannot deal with the change in angle. HectorSLAM produced the worst results of all of the SLAM algorithms because of this. However, if a LIDAR module is obtained and a proper IMU is incorporated into the project, HectorSLAM could still be a viable method of performing mapping.

6.5.3. RGB-D SLAM

VisualSLAM algorithms like RGB-D SLAM, while being very powerful, require large amounts of computational resources in order to properly function. Unfortunately, RGB-D SLAM could not properly run on the Raspberry Pi. The algorithm took up nearly all of the system resources and resulted in the Raspberry Pi shutting down, which

makes it difficult to obtain results. Normally, visual SLAM algorithms are run on computing clusters due to the amount of resources they need to process the data going through. Many Visual SLAM algorithms keep a database of images/features that were obtained from previous frames and process them heavily in order to obtain the most prominent features/landmarks. The Raspberry Pi GPU and CPU is not powerful enough to handle the amount of computation and data lookup that is associated with running these packages.

6.5.4. BreezySLAM

The results from BreezySLAM (Figure 31) were not very different from what was obtained by the GMapping package. The BreezySLAM algorithm did not deal with the robot rotating any better than the GMapping package did.This is most likely because both algorithms work in the same way and both use particle filters; they vary in how they discard older data and points with low probabilities, so better results were not entirely expected. However, the BreezySLAM package required less computational resources and offered more functionality. More modifications can be made to the BreezySLAM algorithm than the GMapping package, which makes it an area where significant future work can be done. Various filters and configuration changes can be made to BreezySLAM that might increase overall mapping results.

6.5.5. Navigation

Due to the functionality that was developed into the navigation stack, the robot is capable of autonomously navigating its environment. The robot will navigate around a map until there are no frontiers left, which indicates that the entire map was explored. However, attempting to navigate an entire room autonomously will take a significant amount of time. Navigating a small, closed environment like the one depicted in the GMapping experiment could take up to 20 minutes. Therefore, navigating a large room could take up to an hour. In addition, there are times when the robot can enter a locked state. If the robot moves into an unknown area or outside the bounds of the known nodes, then it will not be able to calculate its current position or its current position won't be a known node in the global list of nodes. Because of this, it may not be able to path from the robot current position as it will be undefined. This error can only occur if the robot moves too far into an unknown area when it is traveling to a frontier node. Besides this error, the navigation functionality of the robot is sufficient for mapping purposes and will be useful for further work in mapping.

6.6. Control Interface

6.6.1. Raspberry Pi to Arduino Communications

In order to send commands between the Raspberry Pi and the Arduino, a USB serial connection was made to facilitate communication. The Raspberry Pi and Arduino were connected through a standard USB connection. From its serial port, the Raspberry Pi would transmit messages to the Arduino. The Arduino would wait until it had received a message in its buffer, and then execute a case based on that input. The Arduino had various commands that it would execute based on the input such as homing the robot, clearing the encoders, sitting up, or executing a gait. Serial communications between the Raspberry Pi, Arduino and the computer made testing easier and more robust.

6.6.2. Android App Control

As a means of controlling the robot when it was untethered, an Android app was designed to relay messages to the robot. Since the robot would be walking, a standard USB connection to the robot would not work. Therefore, a mechanism for controlling the robot was needed. Due to the capabilities of the Raspberry Pi, there were two options that were explored. The first option was connecting the Android app to the Raspberry Pi through a Wi-Fi connection. The Raspberry Pi is able to act as a router, which means the Android device could connect to the network and send standard messages over a Wi-Fi connection. Additionally, the Raspberry Pi has a Bluetooth module. It was decided that the Android app would communicate using Bluetooth due to the fact that transmitting a Wi-Fi signal on the Raspberry Pi has a short range and requires purchasing an external component.

With the communication protocol decided, the Android app was designed. On startup, the app would ask to connect to a nearby Bluetooth device. Once the device was paired, the Android app could begin to transmit messages. Several buttons with descriptive text such as “Forward” and “Back” were created and added to the main activity of the app. When clicked, these buttons transmitted a Bluetooth message to the Raspberry Pi. The Raspberry Pi would then compare the incoming message and compare it to a list of stored messages and execute the proper function that corresponded to the message. This process was useful in communicating with the Arduino. If a message that dealt with moving the robot was received, the Raspberry Pi would transmit a message over serial communications to the Arduino to begin movement, since the Arduino oversaw all of the low level motor controls.

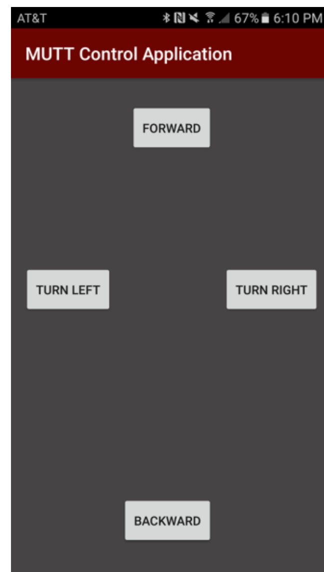


Figure 38: Initial Android App Design

6.6.3. Voice Control through Alexa

The integration of Alexa was a huge contributing factor to the overall functionality of the robot. By integrating Alexa onto the robot, communication between the user and the robot became a lot easier. As mentioned before, the main method of communication was the Android app. The app could tell the robot to startup and start executing its functionality after connecting through Bluetooth and clicking the right buttons. Integrating Alexa onto the device cuts out that portion of the process and allows the user to directly control the robot without having to install an app. Voice control removes a significant amount of overhead both from the user and the developer. From the user, It removes the need to have a smartphone, have an Android device, and have to navigate an app. Speaking is much more natural and intuitive to the end user. On the development side, new functionality can be added quickly by updating the skill's interaction model and utterances, and by adding new conditions to the input parser on the Raspberry Pi, which is significantly easier and less time consuming than having to modify an Android app, recompile, and deploy it.

In terms of controlling the robot's motion, the voice control performed well. It properly parsed the input given to the Alexa device and passed it through the communication pipeline to the Raspberry Pi. Because of the utterances and interaction model, many

different phrases can be given that all produce the same output from the Raspberry Pi. This is very useful because it makes it easier to engage with the user and doesn't require them to follow a strict format when they interact with the user. Overall, the implementation of the voice control was beneficial and simplified the overall control process in ways that couldn't have been done through the Android app.

7. Future Work

7.1. Motion Planning

Updating the scripts that were used to generate the trajectory files would be beneficial to the end user when designing gaits. Although the Matlab script made it easy to visualize the trajectory that was being developed, transforming between the Matlab visualization and the robot movements was not entirely accurate. There are issues with how the robot interprets the Matlab angles that were generated, and there are various variables and equations that can be modified to obtain more accurate results. Specifically, future work in this area should focus on verifying proper transforms between the cam angles generated and the movement of the lower limbs.

Also important in this area is the user interface for the motion planner. The challenging nature of programming new gaits for the robot limits the effectiveness of the robot and the system. Future work should take time to make a more advanced motion planner, in Matlab or otherwise. Important features to include are: Inverse kinematic solvers for both the individual legs and the platform whole, graphical input for the robot motion, equation driven input for robot motion, and direct real-time control of the robot.
Likewise, there exist motion planners built for ROS and other environments that interface with Unified Robot Description Format file. Generating a URDF and interfacing with more advanced motion planners such as Klampt or OpenRave would be a key development to more effectively control the robot.

7.2. Actuation

Replacing the motors and motor controllers used to actuate the upper portion of the leg are a top priority for any future work. Choosing a model of motor that is a bit sturdier than the Vex 393s is a must, as a major problem with the Vex brand motors was the fragility of the internal transmission. Replacing these motors is as simple as redesigning the quarter panels to fit a different model of motor and reprinting them from

the updated models.

Furthermore, replacing the motor controllers is a must, as the ability of the Vex 29's (while an improvement from the Pololu MD10a's) was not quite consistent enough to provide the motors with the current they needed. An excellent replacement is possibly the Fergelli Linear Actuator controller, which includes inputs for quadrature encoders. These controllers are rated for 20amps continuous and so should be well matched to provide the required energy. The ability of these controllers to interpret quadrature encoders is also a boon to the replacement of the motors. The Vex encoder modules were found to be fragile devices prone to damage on an active motor electrical system. Using more standard quadrature encoders integrated with these controllers would produce a more robust system

7.3. Mapping

Due to the robot not being able to move consistently, full stack mapping on the robot alone was not able to be tested. We had to rely on odometry data from the Turtlebot to map an environment. However, due to all of the configuration that was done on the Raspberry Pi, the robot should have the capability to perform different mapping algorithms. In order to do this however, some data needs to be integrated into the odometry topics that the robot broadcasts over ROS. Most of the mapping packages require odometry data to return accurate results. Due to the mechanical nature of the robot, it is difficult to estimate exactly how far the robot has moved without integrating all of the encoder data and applying equations of motion. An alternative to doing this would be to simply estimate the distance that a gait travels and broadcast that distance whenever the robot finishes executing one full cycle of the gait. The data should be more than sufficient when running the various packages and will supplement the algorithms when the laser scans fail.

Additional future work could also revolve around implementing new filters and improving upon BreezySLAM. BreezySLAM is very modular and allows for many changes to be made, and so future students could devise new filters that better deal with the problems that the robot currently has in regards to turning.

7.4. Voice Control

With Alexa constantly growing and becoming more popular, there is significant amount of work that can be done in this area. Voice control opens up various areas of

Human-Robot Interaction beyond the navigation purposes that were described in this paper. With Alexa on-board the robot, the user can now verbally communicate with the robot and talk to it like it would a dog and the robot could respond appropriately. If for example, the user asked the robot to bark, the robot could in fact perform that functionality if coded in. In addition, the robot could be tasked with other functionality like settings alarms, reminding the user to do things, playing music, etc. This makes the robot more personal to the user and makes it more than just a machine. Future work in fixing the movement of the robot and expansion of the robot's language capabilities could result in a robot that has significant meaning and use to the user.

7.5. Mechanical Design

An interesting use of this project in the future may not only be in robotic development, but also for the development of basic design and analysis skills, focusing on its mechanical systems. This robot required the development of custom kinematic and dynamic equations and could thereby be used as a helpful teaching aid in coursework to develop student's analytical skills. Furthermore, the design of the system could be evaluated to determine if the design can be simplified to include less fasteners, more space for electronic components, or less useless print material. Due to the nature of our project, it present numerous opportunities for students in multiple disciplines to develop new skills with regard to robotic systems.

8. Project Insights

8.1. Mechanical Design

During the design and production stages of our printed parts, we found proper clearance to be of paramount importance. Improperly toleranced parts needed to filed or modified by other means to reduce friction at moving joints. This became difficult, however, since we used multiple 3D printers to produce the parts for our robot, and each printer had their own tolerance capabilities. To avoid this, it may have been wise to make small test prints on each machine to observe their tolerances.

Furthermore, the use of standoffs for mounting electrical components, when applied properly, was more useful than originally thought. They not only helped secure our electrical components, but by raising the electronics, they created path through which wires could be run and they acted as securement points to hold wires in place. This

helped to free up space when assembling the robot.

Finally, we found it was important to design above our calculated limits. After determining our limits for torque or motion as discussed in Sections 3.3.1 and 3.3.2, we should have given ourselves a two or three times factor of safety to ensure that our chosen actuators were sufficient for running for an extended period of time. Finally, the plastic gears used both within and in conjunction with the motors failed under the applied load of our robot, and therefore alternative gearing should have been determined.

8.2. Electrical Design

This team underestimated the value of good motor controllers. Our attempts to construct a system without motor controllers designed and built for the power we'd be putting through them resulted in a crippled robot. While a priority on this project was to develop a robot that could be made for significantly less than similar quadrupedal competitors, spending more money on the motor controllers may have brought the project to further fruition than making the attempt with underrated components.

The use of quick connection points between the different components was an excellent decision. Two recommendations for future projects learned herein, reduce the number of connectors even further, and make the securements on the end not crimped into a plastic connector more secure. On the first point it was found that even as few as four connectors was a challenge to connect and disconnect each time the robot was disassembled. There is a balance to be struck between modularity and convenience here, and perhaps one connector per half (split lengthwise) would be more effective. On the second, we used screw terminals on our main breakout to terminate the looms onto the board. These screw terminals consistently damaged the wires to the point of breaking on multiple occasions. A more effective solution might mean using ferrules on the cable ends to protect the stranded wire (our advisor suggested this at the beginning of the project, and we forewent his recommendations to our own demise).

Another successful application of quick connectors was the blue punchdown connectors used to bridge the Arduino and the main breakout. The connectors removed the need to design and construct a shield for the Arduino, but future teams might also consider swapping an offboard breakout panel for a shield (or develop a shield to connect the two with some form of cable). A shield offers similar modularity for the system (if

restricting to a specific form factor of board) while also securing the attachments more thoroughly.

One surprise on this project was the successful application of the Arduino microcontroller. Against the recommendations of our advising team we went forward constructing the project around the Arduino, and to the surprise of perhaps both of us, after a short adoption period it performed admirably. There were hardly any issues with performance in using the Arduino (even after swapping the board for a less powerful model), and programming the device was specifically easy.

8.3. Software Design

8.3.1. Arduino Software

The software design on the Arduino system worked very well for what it was designed for. The design followed an object oriented approach in order to keep the software simple. The most complex part of the software was creation and execution of gaits. Each gait that was created was placed in its own header file. In the file was a series of constants such as size and an array of integers. Each row in the array contained 8 values, which were the angles for each servo/motor. These values could either be generated by hand or exported from the Matlab script that was created to generate trajectories.

Separation of the gaits was crucial as some of the gaits became exceptionally large, and containing them all within one aggregate file would make things unreadable and would decrease overall efficiency when trying to code in that file. Instead, using the declarations that were made in the header files, trajectory objects that referenced the gait files were created. These trajectory objects read in all the data and converted them to frames. A frame object contained all of the angles for the robot at a specific time slice. When the robot executed a trajectory, it was really executing a series of frames over a given duration. This approach worked well and allows for a dynamic way of creating trajectories. By adding to or taking away rows in the gait files, the time duration of the gaits can be modified to fit within specific application requirements.

An interesting aspect of the software was the size of it in terms of memory space. Previous projects on the Arduino boards did not require too much memory space as the programs being written were rather small. Because the gaits were so large, the software

had to be written and stored in different ways (Flash memory as opposed to SRAM) which was an interesting occurrence because memory management was a topic that was taught but until this point was never applied because it was not needed.

8.3.2. Raspberry Pi Software

The software on the Raspberry Pi worked exceedingly well in conjunction with the Arduino system. Many of the scripts that were written for the Raspberry Pi dealt with communication between the Android app and the Arduino microcontroller. It was interesting to see how well it could execute multiple scripts at once that were quite intensive on their own. The Raspberry Pi was running communication, computer vision, and its own OS operations all at once with no delay. However, there was a significant amount of work needed to get everything working together. In order to start all of the needed scripts, several launcher shell files had to be written to launch all of the programs. In order to get functionality like Bluetooth or ROS, several packages and libraries had to be installed. This required a substantial amount of overhead work to be performed, which can be very tedious and frustrating to get right, especially for huge systems like ROS. Despite the substantial setup, this portion of the project gave a significant amount of insight into how embedded computers work and their strengths and limitations, as well as ways to modify and push them to operate according to the necessary standards.

9. Conclusion

Solutions for undergraduate robotics research fall into one of two categories: too simple to be of merit, or too expensive to be available to novice roboticists. Our project sought to eliminate both of these barriers with an inexpensive entry to legged research. We designed and constructed a robot whose programming interface is simple and available to all, and whose hardware was readily reconstructed from inexpensive components. This robot integrated the ability to sense, think, and act; a demonstration of its ability as a robotic platform. Through this challenge, we demonstrated the effectiveness not only of modern low-cost computation hardware, but also the ability to develop an accessible solution to a problem typically reserved for well-funded research laboratories. Future teams wishing to develop such a system should be able to improve on our design significantly, advancing this robot further.

Works Cited

Ackerman, Evan. “Boston Dynamics’ New SpotMini Is All Electric, Agile, and Has a Capable Face-Arm” IEEE Spectrum. 2016. Web. 06 Oct 2016

"Boston Dynamics: Dedicated to the Science and Art of How Things Move”.Boston Dynamics, 2016. Web. 06 Oct. 2016.

Boston Dynamics. "BigDog - The Most Advanced Rough-Terrain Robot on Earth." Boston Dynamics. 2016. Web. 06 Oct. 2016

Brooks, Rodney A. "How to build complete creatures rather than isolated cognitive simulators." *Architectures for intelligence* (1991): 225-239.

Castner, Brian. "The Exclusive Inside Story of the Boston Bomb Squad’s Defining Day." Wired.com. Conde Nast Digital, 25 Oct. 2013. Web. 05 Oct. 2016.

Department of Computer Science: University of Rochester. "Structural Materials for Robots." University of Rochester: Department of Computer Science. n.d. Web. 06 Oct. 2016.

Diaz, Jesus. "Robot Horse Gets First Taste of Real-world Action with the US Marines." Sploid. Gizmodo, 14 July 2014. Web. 06 Oct. 2016.

Kenneally, Gavin, De, Avik, and Koditschek, D. E., "Design Principles for a Family of Direct-Drive Legged Robots", IEEE Robotics and Automation Letters 1(2), 900-907. January 2016.

Ghost Robotics. "Ghost Minitaur." Ghost Robotics. 2016. Web. 06 Oct. 2016

"Ghost | Revolutionizing Legged Robotics -Unmanned Ground Vehicles." *Ghost Robotics*. Ghost Robotics. Web. 20 Mar. 2017.

Hibbeler, R.C. "Engineering Mechanics: Dynamics." Pearson, 2016. Print.

Higgins, Tim. "Google’s Self-Driving Car Program Odometer Reaches 2 Million Miles." The Wall Street Journal. Dow Jones & Company, Inc., 5 Oct. 2016. Web. 5 Oct. 2016.

Hollerbach, John M., Hunter, Ian W., Ballantyne, John. "A Comparative Analysis of Actuator Technologies for Robotics." MIT. 1992. Web. 06 Oct. 2016.

Jayakody, Senadheera. "Basic Facts to Consider When Selecting a Material for a Particular Design." brighthubengineering.com. 2011. Web. 06 Oct. 2016.

Murray, Richard, Li, Zexiang & Sastry, S. "A Mathematical Introduction to Robotic Manipulation." Caltech.edu. 1994. Web. 06 Oct. 2016.

Niku, Saeed B. "Introduction to Robotics: Analysis, Control, Applications." John Wiley & Sons, 2011. Web. 06 Oct. 2016.

Norton, Robert L. "Design of Machinery: An Introduction to the Synthesis and Analysis of Mechanisms and Machines." McGraw-Hill, 2012. Print.

Raibert, Marc H. "LEGGED ROBOTS." Communications of the ACM 29.6 (1986): 499-514. Http://www.sci.brooklyn.cuny.edu/. City University of New York. Web. 5 Oct. 2016.

Schaller, Robert R. "Moore's law: past, present and future." IEEE spectrum 34.6 (1997): 52-59.

"Sony Aibo: Just Beautiful." Sony Aibo. Sony Aibo Tribute Site, 2016. Web. 06 Oct. 2016.

Spong, Mark W., Hutchinson, Seth & Vidyasagar, M. "Robot Dynamics and Control." Northwestern.edu. 2004. Web. 06 Oct. 2016.

Wieber, Pierre-Brice, Russ Tedrake, and Scott Kuindersma. "Modeling and Control of Legged Robots." Springer Handbook of Robotics. By Bruno Siciliano and Oussama Khatib. Cham: Springer International, 2016. 1203-234. Print.

Will Bosworth, Sangbae Kim and Neville Hogan, "The MIT super mini cheetah: A small low-cost quadrupedal robot for dynamic locomotion", Safety Security and Rescue Robotics (SSRR) 2015 IEEE International Symposium on, pp. 1-8, 2015.