Project Number: MXG-WS20

WPI Mascot Robot

A Major Qualifying Project Report
submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the

Degree of Bachelor of Mechanical Engineering
Degree of Bachelor of Robotics Engineering
Degree of Bachelor of Computer Science
By:

Jonathan Chang (Computer Science)

Connor Dietz (Mechanical Engineering)

Treksh Marwaha (Robotics Engineering)

Griffin Roth (Robotics Engineering)

Date: May 18, 2020

Approved:

_____
Professor Michael A. Gennert, Major Advisor

_____
Professor Gillian Smith, Major Advisor

_____
Professor Holly Ault, Major Advisor

This report represents the work of one or more WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review.

Abstract

The WPI Robot Mascot MQP is designing and building a robot goat to be a companion mascot to Gompei. Robotics is a signature program of WPI, and this MQP seeks to further represent the iconic robotics work done by students and faculty. It would be impossible in one MQP to develop a mascot robot that reflects the impressive robotics achievements at WPI. This preliminary MQP involved design and prototyping work for 5-DoF head and neck assembly. This includes work on the mechanical, electrical, and software systems of the robot, as well as the initial requirements and project planning done in collaboration with various stakeholders at WPI. Future MQP teams will further develop the robot, and it will serve as a continuously evolving platform to showcase the traditions and experiences at WPI.

Acknowledgements

Though our work on this MQP is only the beginning of a mascot robot, the unending support we have received from our friends, families, professors, and mentors has us confident that we will return to campus one day to find a lovable robot goat running around campus and inviting everyone to see what WPI students can achieve.

We would like to acknowledge the WPI community for embracing our idea on a whim, for making it easy to talk with students, faculty and professors throughout campus, for never hesitating to make time in their busy schedules to give us the help we needed, and for showing us how to create something real with our education.

We would like to acknowledge our advisors for their dedication to our learning and success, for mentoring us in leadership and interdisciplinary problem solving, and for giving us the resources to struggle with confidence throughout this project.

Lastly, we would like to acknowledge the countless individuals that gave us invaluable inspiration and feedback, that made connections, that provided resources, and of course, those that put up with late nights (well, early mornings, really).

To everyone that had a hand in this project, we thank you for being a part of this unique culminating experience with us. We think it is safe to say that we could not have even started it without you.

Table of Contents

Authorship

Section	Primary Author	Secondary Author	Editor
Abstract	All	-	-
Acknowledgements	Connor Dietz	-	-
Author's Note	Connor Dietz	Griffin Roth Jonathan Chang	-
1	Griffin Roth	Jonathan Chang	Connor Dietz
2	Griffin Roth	Jonathan Chang	-
2.1	Griffin Roth	Jonathan Chang	Treksh Marwaha
2.2	Connor Dietz	Griffin Roth	Treksh Marwaha
2.3	Griffin Roth	Connor Dietz	-
2.4	Griffin Roth	Connor Dietz	-
3	Connor Dietz	Griffin Roth	Jonathan Chang
3.1	Connor Dietz	Griffin Roth	Jonathan Chang Connor Dietz
3.2	Jonathan Chang	-	-
3.3	Connor Dietz Jonathan Chang	-	Connor Dietz
3.4	Connor Dietz	Griffin Roth	-
4	Connor Dietz Griffin Roth Jonathan Chang Treksh Marwaha	-	-
4.1	Connor Dietz		
4.1.1	Connor Dietz		
4.1.2	Connor Dietz		
4.2	Griffin Roth Treksh Marwaha	-	Jonathan Chang
4.2.1	Griffin Roth	Connor Dietz	Jonathan Chang Connor Dietz
4.2.2	Griffin Roth	Connor Dietz	Jonathan Chang
4.2.3	Griffin Roth	Treksh Marwaha	Jonathan Chang
4.2.4	Griffin Roth	Treksh Marwaha	Jonathan Chang
4.2.5	Griffin Roth	-	Jonathan Chang Treksh Marwaha
4.2.6	Griffin Roth	-	Treksh Marwaha
4.2.7	Griffin Roth	-	Jonathan Chang
4.2.8	Griffin Roth	-	Jonathan Chang
4.3	Jonathan Chang Treksh Marwaha	-	Griffin Roth
4.3.1	Connor Dietz	Jonathan Chang	Griffin Roth
4.3.2	Jonathan Chang Treksh Marwaha	-	Griffin Roth Connor Dietz
4.3.2.1	Jonathan Chang	-	Griffin Roth
4.3.2.2	Jonathan Chang	Treksh Marwaha	Griffin Roth
4.3.2.3	Treksh Marwaha	-	Griffin Roth
4.3.3	Treksh Marwaha	-	Connor Dietz
4.3.4	Jonathan Chang	Treksh Marwaha	Griffin Roth Connor Dietz
4.3.5	Jonathan Chang	-	Griffin Roth

			Connor Dietz
4.3.6	Jonathan Chang	-	Griffin Roth
5	-	-	-
5.1	Connor Dietz		
5.2	Connor Dietz	Griffin Roth Treksh Marwaha	Jonathan Chang
5.3	Jonathan Chang Treksh Marwaha	-	-
5.3.1	Jonathan Chang Treksh Marwaha	-	Griffin Roth
5.3.2	Jonathan Chang	-	Griffin Roth
5.3.3	Jonathan Chang Treksh Marwaha	-	-
5.3.3.1	Treksh Marwaha	-	Treksh Marwaha
5.3.3.2	Jonathan Chang	-	Griffin Roth
5.3.4	Jonathan Chang	-	Griffin Roth
6	Connor Dietz Griffin Roth Jonathan Chang Treksh Marwaha	-	Treksh Marwaha
6.1	Jonathan Chang Connor Dietz	Treksh Marwaha	Treksh Marwaha
6.2	Connor Dietz	-	-
6.3	Griffin Roth	-	Treksh Marwaha
6.4	Jonathan Chang Treksh Marwaha	-	Griffin Roth
Report compilation and formatting		Connor Dietz	

Authorship Role Explanations:

Primary Author*: The group member(s) responsible for writing most of the section content.*

Secondary Author*: The group member(s) responsible for adding significant section content.*

Editor*: The group member(s) responsible for making spelling and/or grammar corrections, or content suggestions (but not modifications).*

List of Figures

List of Tables

List of Equations

List of Abbreviations

Abbreviation	Definition
GB	Gigabyte
dB	Decibel
V	Volts
WPI	Worcester Polytechnic Institute
cm	Centimeter
GHZ	Gigahertz
DOF	Degrees of Freedom
FOV	Field of View
FPS	Frames Per Second
AI	Artificial Intelligence
mm	Millimeter
Op-amp	Operational Amplifier
OS	Operating System
FSM	Finite State Machine

Authors' Note

This report is not the final iteration for the WPI Mascot Robot MQP. Some members of the team will be continuing the project in subsequent term(s). As such, this MQP report reflects the work done up to the end of the fourth project term (D20). The work herein has been presented such that, to the extent possible, its content is consistent in isolation. However, some of the team is still focused primarily on project work, so there may still be section stubs that are yet to be completed. Where appropriate, some sections note their incomplete status if it is not self-evident. Note that this is also not the first version of the report published for some team members' completion of degree requirements.

COVID-19 Response

This MQP, like many others, was affected by the coronavirus (COVID-19) pandemic. This segment of the project was originally intended for the integration, testing, and validation of the work produced during the A19, B19 and C20 project terms. Unfortunately, health safety concerns as well as most of the team traveling home from WPI meant access to project materials and resources was highly limited. To cope with this, adjustments were made to the term goals as well as the intended deliverables. Relevant sections in this report to affected project elements include a statement on how they were affected, and, if applicable, how they will continue to be handled in subsequent project term(s).

Though the impact to the project was large, the team saw an opportunity to frame the experience as learning to work in a horizontal integration environment. Reviewing materials or transferring work was no longer as simple as walking to an office. In the same way that design firms must work with overseas manufacturers and suppliers, collaboration had to be done remotely. The team members learned to effectively communicate ideas asynchronously and developed their ability to work with unpredictable schedules.

1 | Introduction

Robotics is a signature program of WPI, however, there is still an opportunity to further integrate robotics into WPI's image. While it is not the only important part of WPI's identity, the robotics work its students and faculty produce are iconic and recognizable to people both in and out of STEM fields. The goal of this project is to design a prototype of a mascot robot for WPI that anyone can interact with to see and feel what WPI is all about.

However, WPI already has a mascot—Gompei the Goat. Gompei is ingrained in the WPI tradition and loved by many, so it would be a huge loss to replace him. The aim of this project is not to replace Gompei, but to add to the Gompei experience with a little robot flair, as is WPI tradition. This robotic mascot will be able to operate in a wider variety of circumstances such as a semi-permanent display, being part of admissions tours, and events during unsuitable weather.

The current design of the robot sees the robot taking the form of a goat, with proportions like those of a British Alpine goat. The robot will have 2 modes: teleoperated and autonomous. In the teleoperated mode, a user will control the robot with a remote control, capable of making the robot move its head and limbs. In the autonomous mode, the robot will stand in place and be capable of interacting with people. It will be able to track the movement of hands, turn its head to face people talking to it, react to being pet, and make noises in response to what is happening around it. However, this project will not be creating the full robot. A full robot of this scale requires an immense amount of work, not possible with the given time. Instead, this project will serve as a proof-of-concept for a mascot robot that future projects may build upon and complete. This project will primarily focus on designing the head and neck mechanisms, as well as the accompanying AI and basic circuitry for motor control and sensors.

Over the course of A-Term, this team spoke with potential stakeholders in the project, including Admissions, Athletics, and the SAS to get their thoughts on what the robot will be used for and what it should be capable of doing. Using the requirements created from those responses, a character design was drafted and

finalized, and initial designs for the mechanics and software of the robot began. During B-Term, the requirements and designs were further refined. Prototype versions of both the mechanisms as well as the software were produced, analyzed, and integrated. Requirements for sensors were created, and applicable sensors were identified. In C-Term, the implementation details for the mechanisms and software were determined. The first version of the software for the robot was written with the functionality to actuate physical motors to dynamically created positions; production of physical mechanisms also began with additional focus being put into electrical design. This report will highlight what options were considered, analyzed, and selected for the first version of the mascot robot.

2 | Background

2.1 | Mascots

Mascots are an important part of organizational branding. A strong brand communicates an image that builds trust and loyalty with its consumers. Mascots help create an emotional bond between the customer and the brand, as they now have an anthropomorphized character they can associate with the brand [1]. This in turn helps engage customers with that brand. Of course, mascots are also an important part of the brand of schools. Many high schools and colleges have mascots that help people and alumni identify with the school. For example, WPI has Gompei the goat, Penn State has the Nittany Lion, and MIT has Tim the beaver. These mascots appear in their respective school's marketing, merchandising, and at events as costumes worn by people to bring the mascot to life.

Figure 2.1 2020 Tokyo Olympics mascot robots Miraitowa (center right) and Someity (center left).

Mascot robots, in comparison, are a new concept. The only public mascot robots are the robot versions of Miraitowa and Someity, the mascots for the 2020 Tokyo Summer Olympics designed by Toyota (Figure 2.1) [2, Fig. 1]. These mascot robots take the form of small, toy-like robots that will welcome athletes and guests to venues, as well as interact with children [2]. Although these mascot robots do not replace the original mascots, they provide an opportunity for their creators to show off their technical prowess and interact with the audience in new ways. Due to the

lack of other mascot robots, there is a large design space open for new mascot robots with different capabilities and appearances.

2.2 | Human Interactive Robots

One design space of robots that is currently seeing a lot of experimentation is human interactive robots. Several robots today are being built with the intention of interacting with humans. These range from companion robots such as Jibo and Blue Frog Robotics' Buddy to robots used for therapy and understanding human-robot interactions. These robots interact with people in various ways, some respond to audio cues, others move or react to stimuli; some even show emotions. As the field of robotics advances, we will inevitably see human-robot interactions become more frequent. One important development that we will likely see as social robots advance, is the ability to display empathy. Empathy requires one to be able to "perceive, understand, and feel the emotional state of others" [3]. A robot with this ability would need to be able to understand the context surrounding a situation, then decide on what to do.

2.3 | Goat Anatomy

Before any mechanisms can be researched or design work done, it is important to first understand the anatomy of the animal that this robot will be based around—that being a goat. The anatomy of any animal can be broken down into multiple subsystems [4]. When studying the model animal anatomy to inform the design of a robot, two obvious systems to start with are the skeletal and muscular system. Together, these biological systems form the foundational structure of the animal and provide a means for articulating it [4], [5].

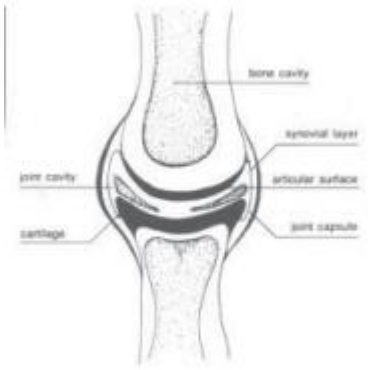


Figure 2.2 Anatomy of a Goat Knee

As with most animals, goats have joints with both single and multiple degrees of freedom (DoF). Unlike mechanical systems, these joints typically do not have consistent instant centers. For example, the knee joint appears to simply be a single DoF joint with a fixed point of rotation. However, the rotational axis of this joint is variable due to the structure and alignment of the fibula, tibia, and the many support bones (Figure 2.2) [4], [5]. Evidently, in order to translate these biological systems into mechanical ones, simplifications will need to be made. These simplifications are the functional degrees of freedom [6].

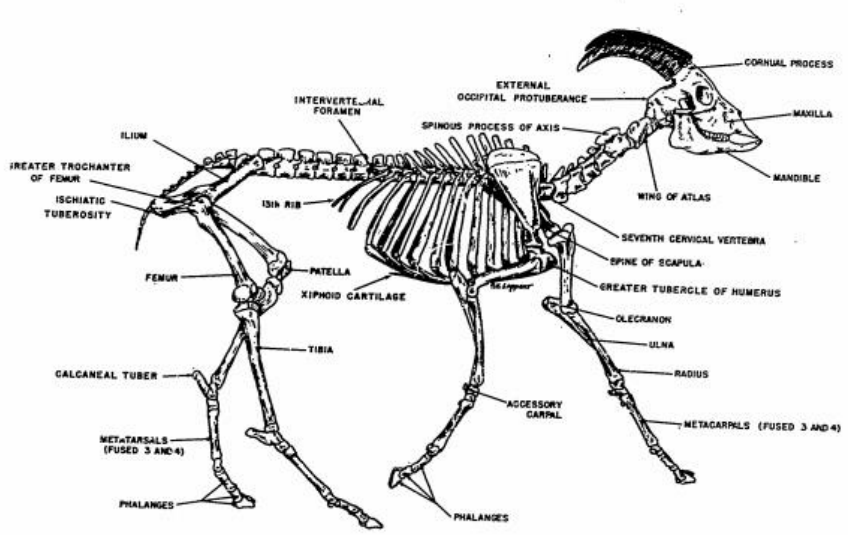


Figure 2.3 Goat Skeletal Structure

For this project, the important joints to examine are the ones that influence head and neck movement. Goats, like most other animals, have six degrees of freedom of positioning capability for their head relative to their shoulders [6], [7],

[8], [9]. Functionally, this is accomplished through their 7 cervical (neck) vertebrae (Figure 2.3) [4, Fig. 1] [4], [5]. This chain of vertebrae is constructed from seven 3DOF joints chained together by the neck ligaments. Some of these vertebrae are underactuated, and instead serve as intermediate pivot points along the chain. Notably, these vertebrae tend to be fully articulated about the transverse and sagittal axes, but minimally about the longitudinal axis [6], [9]. Rotation about the sagittal, transverse and longitudinal axes corresponds to a goat 'looking' left/right, up/down, and tilting, respectively.

2.4 | Animatronics

With the anatomy of a goat established as a reference point, considerations for how to translate that into a mechanical system can be explored. Traditional robotic systems do not always have the requirement of mimicking or resembling biological creatures. It makes sense, then, to turn towards the field of animatronics, and to understand how they can simulate organic movements.

Animatronics is effectively a form of puppetry, using mechanisms and robotics to simulate a living being moving and sometimes interacting with the world around it. Animatronics often see use in filmmaking, theme parks, and other forms of entertainment. One of the benefits of animatronics is the life-like appearance they can take. Instead of animating something using computer graphics, the animatronics look real in comparison [10]. In animatronics, a lot of work and care goes into the controls and mechanisms to make the subject and its movements as realistic as possible.

Mechanical design for animatronics brings with its new design considerations compared to traditional robotics. One of the biggest differences in requirements between robotics and animatronics is that animatronics usually do not need to physically interact with users or their surroundings. This means that not only can mechanisms be lightweight, but less overall system rigidity is required. Additionally, less power is generally required to actuate a given join, so mechanisms

can be designed to use complex systems of cables, pulleys and linkages that do not provide much mechanical advantage at the effector [11].

The ability to focus mostly on desired movements and speeds gives more options for actuating joints with multiple degrees of freedom. For example, when designing a neck, there are many routes one can take. If simple, jerking motions are okay, a series of bevel gears driven by motors and servos will suffice. For more intricate motions in smaller applications, the neck can be actuated by two shafts with u-joints attached to servos [15]. For more complex applications, a tentacle mechanism can be used which utilizes a series of cables and u-joints to make the neck move in complex patterns. When one cable shortens, other cables lengthen, bending the neck [16]. These mechanisms do address the problem of creating a neck, however the abilities and range of motion vary between them. Therefore, it is important to always design for the problem at hand, and not for what would be the most versatile.

3 | Project Conceptualization and Planning

3.1 | Outline the Project

The conceptualization of this project stemmed from our previously planned project to develop a robot mascot for the Worcester Red Sox, however because of this, much of the initial project needed to be dedicated to identifying the project goals themselves. After deciding to continue with the robot mascot idea, this time for WPI, we began discussing how to translate it for our new target audience. Though we did have a rough initial plan based on previous planning with our contact at the Worcester Red Sox, we did not have a clear overall goal for the project and needed to outline a plan and a set of goals. To facilitate this planning, we used a set of guidelines laid out in Systems Engineering for Capstone Projects [17] to inform our process. This framework guided us through the process of identifying stakeholders, determining their needs, developing the concept for the robot, translating those needs into requirements, and then developing a robot character and design from them.

The first phase, the stakeholders and needs analysis, ultimately became an ongoing process rather than an initial step. This was particularly important as we were frequently faced with situations where we needed to evaluate our needs regarding previously nonexistent project elements. All the requirements and specifications for this robot are derived from conversations with our stakeholders, namely the WPI Athletics and Admissions departments, as well as our own brainstorming and discussions on usability and behavior. Some of these initial conversations led to similar discussions about unpredictable behavior from the users, particularly children and rowdy college students. Those points laid the foundation for how we would need to think about the rest of the design process. Safety would need to be an integral part of the final product, along with usability and predictability.

We needed to first have conversations with our stakeholders about how they would like to see the robot used, and the types of environments it would be placed

in. While we did not strictly adhere to each step, it was an invaluable starting point to help us formulate our project goals.

Every project requires stakeholders who are interested in using the final product. As this project was conceptualized by the team, there were no initial stakeholders. Therefore, the team had to find and market the project to potential stakeholders that would be interested in seeing the project's success. Potential stakeholders were chosen based on their perceived potential usage for the robot (Table 3.1).

Table 3.1 Stakeholder Identification and Involvement

Stakeholder	Involvement/ Type	Met by …	Rationale
WPI Athletics Department	Direct, Consulted, Informed	Periodic meetings with Dana Harmon (director)	Mascot present at sporting events
WPI Student Body	Indirect, Consulted, Informed,	Informed through social media (future)	Will interact with the robot during events
WPI Robotics Department	Indirect, Consulted, Partner, Informed	Presenting project during project presentation day	Must satisfy the department's requirements for MQPs
WPI Alumni, SAS	Direct, Consulted, Informed, Approver	Compliance with guidelines set for acceptable use of Gompei, Periodic updates with Herd Chair	Will manage the robotic mascot's involvement in events.
Fans of WPI Sports, Programs	Indirect	Shown the robot at events.	Will interact with the robot during events
WPI Admissions	Direct, Engaged	Periodic meetings with Isabella Camasura (Admissions Counselor). Financial Aid.	Will present the robot at certain locations as a display.

Three major stakeholders at WPI were identified- Admissions, Athletics, and the Student Alumni Society (SAS). Each of these stakeholders were identified as possible parties that would be using the robotic mascot in the future. Admissions expressed interest in showcasing the robot during admissions tours as a demonstration of all the different engineering fields at WPI. During meetings, Athletics considered having the robot as an additional moral and spirit source— the robotic mascot was envisioned interacting with the crowd during breaks or posing

whenever the WPI sport team scores. The SAS is the organization responsible for managing the appearance of the Gompei the Goat mascot and would be the organization that would take care of this robotic mascot including the storage, maintenance, and supervision. During the requirement gathering phase, the team met with each of these stakeholders to help determine the functionality requirements of the robot.

3.2 | Stakeholder and Needs Analysis

Table 3.2 Stakeholder Needs

Stakeholder	Needs	Stakeholder Considerations
Athletics	• The system should emote or react during sports game • People can take photos of it (put their arms around it) • Celebratory pose/action on goals, touchdowns, runs, etc. • The system should be relatively easy to transport	The system will be at sporting events, so it should be able to interact with and hype up the crowd. The system will also have to not track debris onto the basketball courts.
Admissions	• The system should interact with visitors and inform them about WPI • The robot could be placed in the upcoming Bartlet Center lobby museum • Visitors can take selfies with it • Robot could have a fake smartphone in one of its hooves that takes selfies	The system will be on display for visitors, so it should be visually appealing and safe for visitors to approach. The system may also be used to inform visitors about WPI.
SAS	• The system should be able to create screaming goat sounds • The system should be able to act as a sound system • The system should be able to launch merchandise • The system should be able emote • The system should have a soft exterior • The system should not speak	SAS will handle the arrangements for the system. They will also have some approval over the appearance and manner of the system.

Each major stakeholder identified a different set of needs based on their expected use (Table 3.1). Athletics pointed out that given the robot would have to be

transported across campus, it was imperative that the robot be easily movable; doors, curbs, elevators, and other obstacles would have to be overcome during transportation. Admissions brought up the idea that people might like posting about the robot on social media and the mascot must have the size and configuration for people to be able to take photos with it. SAS shared a list of guidelines for the Gompei the Goat Mascot which included some needs that were not initially considered. For instance, a mascot should not speak or express any views on any subject other than support for the current sports team.

Table 3.3 Stakeholder Need Priority

Need	Statement	Validation	Priority
Safety	The robot should be safe to approach and operate	Robotics Department, SAS, Admissions	1
Interaction	The robot should be able to react to audio, physical, and visual stimuli	SAS, Admissions, Marketing	1
Photo taking	The robot should be able to take photos of itself with another person.	Admissions, Marketing	2
Movement	The robot should be able to move to another location without external forces	Athletics, SAS	1
Remote Control	The robot should be able to be controlled by external controls	Athletics, SAS	1
Head Movement	The robot should move its head in multiple directions	Admissions, SAS, Athletics,	2
Autonomy	The robot should have an autonomous mode that reacts to external stimuli	Admissions, SAS	2
Visually Appealing	The robot should appear as a cartoonish goat not a robot.	SAS, Admissions, Athletics	2
Waterproofing	The robot should be able to handle some rain exposure	Athletics, SAS	2
Overheating	The robot should not overheat due to internal heating issues	All	1
Replaceable feet	The robot should have replaceable feet to avoid tracking dirt inside buildings	Athletics	2

The initial stakeholders in the list were then used to create a prioritized version that indicates which stakeholders have the greatest stake in the project. From this we could then see which stakeholder needs should be focused on to develop the requirements for the robot. The derived prioritized list of needs (Table 3.3) has information about the need itself, along with a note on which stakeholder it originates from, and an indication of its relative priority. In the table, needs of the highest relative priority are marked at a "1", and secondary priorities at a "2." These rankings were largely subjective and were based on our own intuition and synthesis of the concerns of our stakeholders, as well as our advisors. This list of needs was then used to build a more detailed, formal list of robot requirements that used more technical language (Appendix C).

3.3 | Character Concept

Developing the character of the robot we would be creating was the first real step towards designing the robot itself. After initial meetings with our stakeholders, we gathered our ideas and drafted up some initial designs. At this stage, we also considered how the robot might be manufactured, and what materials could be used for the outer 'skin' of the robot. What we ended up with was a collection of sketches and listed ideas that we then took back to our stakeholders for feedback. While none of the team members proclaimed to be artists, we worked with what we had and ultimately created a set of ideas that convinced our stakeholders that this robot could be a potentially interesting addition to the tradition of WPI.

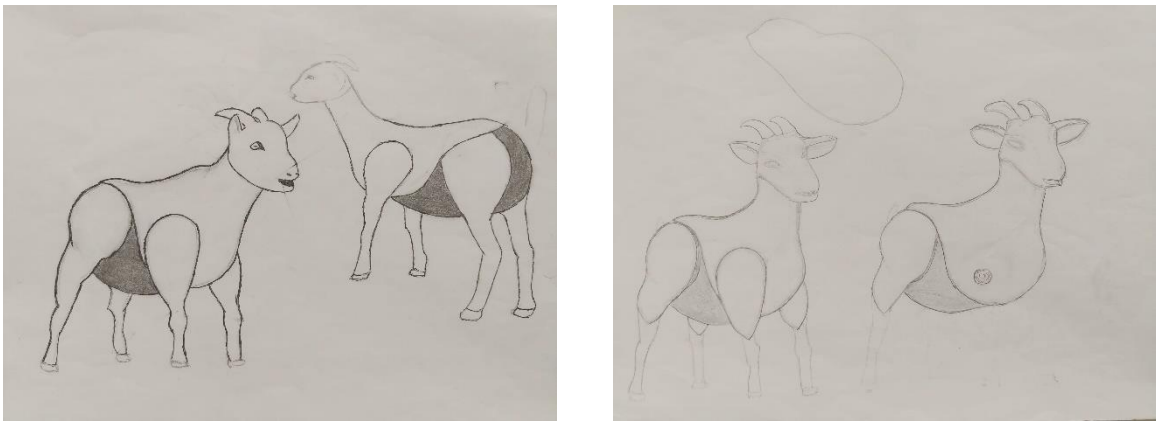


Figure 3.1 Character Sketches of the Robot

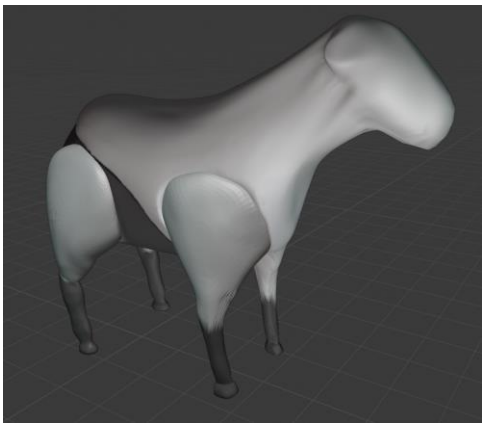


Figure 3.2 Digital character sculpt

The character that was created was Robo-Goat (name pending). It would look and behave similarly to a real goat. Robo-Goat would still be able to appeal to people, without overshadowing Gompei as a character. The process of sketching our results was not easy for us as we did not have much prior experience in freehand drawing. With our sketches, we also created a first version digital sculpt (Figure 3.2) of the character to get a better understanding of what it would look like and how we might go about manufacturing it in the future. The final design (Figure 3.1) we arrived at was based on things we liked from our various initial sketches, and represented what we felt would create an inviting character, but that still maintained a distinctly robotic appearance.

The reasoning for going with this design is it is distinctly a robot; however, it still retains the organic visual characteristics of a goat. Also, and almost more

importantly, it does not exude an uncanny creepiness or appear to be dangerous, which is crucial for inviting spontaneous interaction. We also noted that if something looks dangerous, there's a good chance it probably is, so we avoided classical robot design tropes involving exposed moving components and jagged edges. A cleaner design, on the other hand, appears more visually inviting, and lends itself well to designing for safety.

As mentioned, we also considered the physical materials and manufacturability of the robot at this stage. In this design, the head and upper body will be covered with a thin, continuous silicon sleeve that is placed over a solid under-shell. The hard under shell will both add solid form to the robot, giving it a goat-like appearance, as well as protect the internal components from users, and the users from internal components. The soft, silicone outer sleeve will give the robot its details, as well as a more inviting texture that also protects against injuries at pinch points.

4 | Robot Design

4.1 | Mechanical Design

As the mechanical design of the robot was done concurrently with the software and electrical design, we took an iterative design approach. Particularly, designs needed to accommodate the designed power budget, as well as the sensors on board. Rather than constructing functional prototypes after a round of design was completed, we prototyped elements of the mechanical systems throughout the process. Despite the increased time investment on revisions of physical products, this approach was necessary to perform periodic testing of co-dependent systems. This integrated design strategy ultimately reduced the need to redesign dependent components at various stages of development.

The mechanical design process was facilitated by a set of digital tools as well as rapid prototyping hardware. Two CAD packages were used during this project. The cloud based solid modeling software, Onshape (recently acquired by PTC), was

used for the initial concept designs and prototyping. Onshape was chosen here for its collaborative editing functionality. This always gave each team member access to CAD models and sketches and made working together to quickly prototype designs easy. Dassault Systèmes SolidWorks, along with its stress simulation tools were later used for the final design and analysis. While Onshape is very robust in its current state, it does not offer the same product maturity as SolidWorks, including its analysis suite. Most of the prototyping work for this project was done using FDM 3D printers. Though these machines dramatically reduce time spent iterating on prototypes, 3D printed parts alone could not provide us with the functionality needed. Most prototypes consisted of a combination of 3D printed parts and OEM components such as bearings, fasteners, and actuators. In some cases, these designs were also supplemented with more rigid materials such as extruded aluminum beams.

In most cases, parts were designed with their final materials in mind. The nature of 3D printing meant that we could produce a working prototype of a part eventually intended to be machined out of aluminum, for example, minimizing the design steps required before final production. Due to the coronavirus pandemic, we were not able to reach a point where we could produce final components.

4.1.1 | Initial Design Parameters

The mechanical work required for this robot encompasses a range of design areas, including mechanism design, structural design, as well as the accompanying kinematics and analysis for them. Given the large scope of the overall project, it was necessary for us to focus in on a smaller set of design goals. Furthermore, most of our decisions are further focused on the design of everything from the neck up. Our initial work focused on identifying the key design constraints that would govern the rest of the development of the robot. We outlined the degrees of freedom the neck and head assembly would have, the sizing and scale of the robot itself, the strength requirements of various components, and the speed and acceleration needed for fluid motion of each joint. It was important that we consistently consider a set of common factors to ensure the robot could perform its tasks both safely and reliably.

As such, failure modes of many different types were considered throughout the design process and were driving factors in the decision-making process for the design constraints. The following sections provide details on each of these design decisions.

⊣ Degrees of Freedom

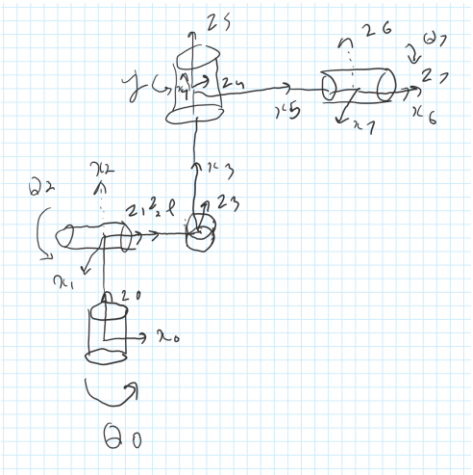


Figure 4.1 Kinematic outline of the neck and head mechanism

The current robot is designed around a 5DoF head and neck assembly along with a base chassis that acts as the frame of the body. The head will be capable of three rotational degrees of freedom about the top of the neck, and the neck will have two rotational degrees of freedom about the shoulders. Figure 4.1 shows a rough kinematic outline of the neck superimposed on an image of a goat. Note that Figure 4.1 depicts a serial kinematic chain, whereas our final design has a parallel joint controlling the head, as detailed later. Currently, our design is focused on everything from the shoulder up. In this way, the robot will be able to look up, down, left, and right in an organic fashion, along with the ability to tilt its head for added expressiveness. Later projects can add additional capabilities to the body and legs.

Though a real goat has many more links and joints in its neck than our robot, replicating them would add a large amount of complexity, increasing development time and introducing many more failure modes. Added complexity further increases the difficulties of designing a moving system intended for use by inexperienced

users. We determined that five degrees of freedom for the neck and head would provide a good balance between simplicity and expressiveness.

⊣ Sizing and Scale

The physical sizing and scale of the mascot robot was a critical early step in the design process. This affects not only the interaction experience, but also relates heavily to decisions that need to be made concerning safety.

Initially we used a real goat as a size reference. This would make the robot roughly the size of a large dog, with its head presenting itself at about waist high on an average person. However, after interacting with several real goats and taking our own measurements, as well as discussing with our stakeholders about how the robot would be used, we determined this to be too small to meet project needs. The robot needed to be large enough such that users would feel as though they are interacting with something substantial. We could not make it too large, however, as we wanted the robot to feel unintimidating and approachable.

Of course, an increase in size also brings an increase in associated costs due to longer moment arms and thus large power requirements of the actuators and strength requirements of the structure. Additionally, we have not yet discussed the relative scaling of the anatomical analogues to our corresponding robot sections. Like how we determined that five degrees of freedom would give us our desired expressiveness, we needed to consider how the relative sizing of the various robot components would affect the character we were creating. Instead of designing with the goal of maintaining accurate relative scaling, we instead focused on ensuring our character was both exaggerated, but still lifelike and recognizable. Taking these design liberties also allowed us to modify the sizing of different components to better handle the loadings we needed to account for, so designing for strength and rigidity would be much easier.

⊣ Strength and Rigidity

In discussions with our stakeholders, it became apparent that our robot would need to be capable of withstanding a fair amount of abuse. Discussions with our stakeholders and our own intuition were used to examine potential issues the mascot robot might encounter from users. For example, the robot will inevitably experience physical disruption from a child. This not only meant that we need to design with structural rigidity and integrity in mind, but also design for safety. Injuring a child is a less than ideal outcome, so we set our safety factor for critical components at 6. More properly, components that may potentially experience additional loading from a user, or components whose failure could result in injury, must have a yield stress of at least 6 times higher than their maximum expected stress under those conditions. This decision was primarily based on recommendation from our mechanical engineering advisor, Professor Holly Ault.

A safety factor of 6 is indeed quite large, so we needed to carefully consider which components needed to be held to that standard. To maintain consistency in design, we needed to outline a set of loading conditions to test various components under. These test conditions needed to be useable for both the designs as well as using the eventual manufactured parts. Furthermore, we needed to maintain consistency between the different tests, and develop a framework that can be used to track results.

⊣ Range of Motion and Joint Speeds

Determining the physical limits of articulation for each joint was crucial to identifying possible mechanical joint designs, and it also determined the expressiveness and what types of behaviors the robot would be capable of exhibiting. Initially, our goal was to simulate the movement of a real goat as closely as possible. In fact, we started by using some of the footage and images collected from our interactions with real goats to measure observed limitations of their ranges of motion. Additionally, we were already aware that convincingly emulating the organic movements of a known animal is difficult. After discussing with our

stakeholders, namely the WPI Athletics and Admissions departments, we decided our goal should instead be to create a caricature of a goat rather than a lifelike representation of one (as detailed earlier in sections 3.2 | and 3.3 |). Now that we no longer needed to accurately represent the movement capabilities of a real goat, we instead decided on motion ranges for each joint that would give us the freedom needed for the expressiveness that we wanted. To do this we simply estimated what kind of motions we thought would be expected, and then added some additional articulation range as a margin of error.

Joint speeds were determined in a similar way of estimation. We used techniques such as moving our arms in a sweeping motion at a speed that felt natural and measuring the time it took to complete it. We also factored in the location of the joint along the kinematic chain and how large the part of the robot it would be moving. In most cases, movements would be completed by more than one joint, which in many cases increases the resultant angular velocity of the head. This meant we could reduce our specified speeds for the upper head joints, which provided benefits such as reduced weight requirements.

4.1.2 | Upper Neck and Head Design

As the entire neck and head assembly of the robot would ultimately create one long moment arm, we decided it best to start the design at the head and work backwards. The neck structure and the joint at the base of the neck must be able to support not only our prescribed test loadings, but also the weight of the various upper components themselves. As stated, the head joint must be capable of three degrees of freedom. To start, we focused on the pitch and yaw motions of the neck. The roll action will be added later. We identified several mechanisms that could potentially fulfill requirements and performed some surface level analysis to narrow down our selection. First, we will discuss some general decision factors used, then we will detail the current design, and finally we will provide information on the other options we considered.

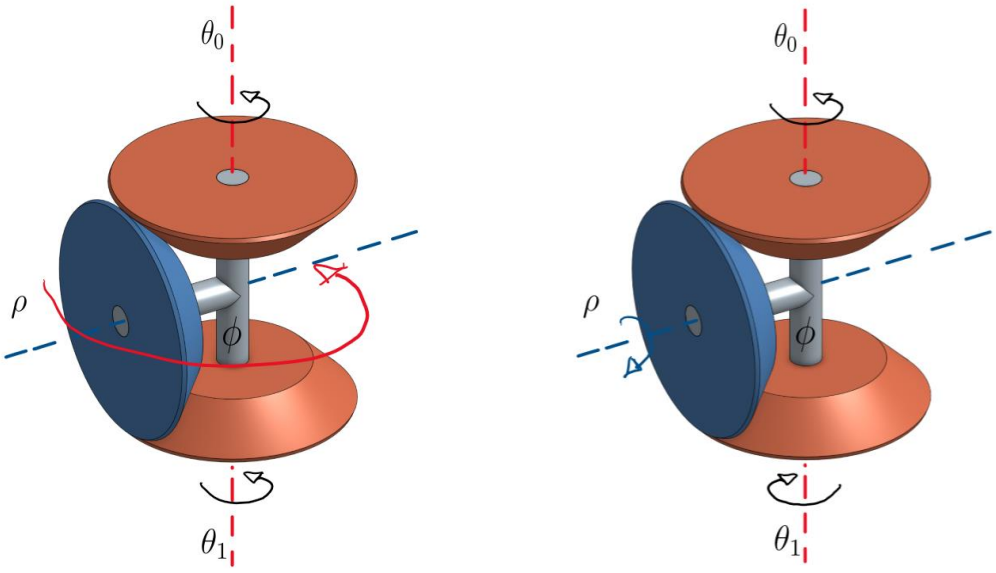
In addition to the loading requirements we determined (pg. 32), we considered mechanism complexity and control requirements when selecting a

suitable design. These impact design difficulty and manufacturability, as well as how computationally expensive the mechanism would be to control. The design complexity would also affect the difficulty of the analysis required to prove the design capable of safely withstanding our prescribed loadings.

To actuate the joint at the base of the head, we decided that a pulley actuated differential mechanism would best suit our requirements. Our decision was made after considering a variety of different multiple degrees of freedom mechanisms and how they would satisfy our requirements. Namely, we focused on selecting a base mechanism that could provide us with our specified articulation angles (see section 4.1.1 |), was possible to design to be reasonably compact, and would successfully meet our loading requirements. With the base mechanism selected, we designed our implementation of it.

⊣ Head Joint Kinematics

At this point in the project, we have two versions of the differential mechanism in use (discussed later in section 5.1 |), however they both operate under the same kinematic principle.



(a) **(b)**

Figure 4.2 Simultaneous (a) and differential (b) input rotation of the differential mechanism

Figure 4.2 illustrates the rotational kinematics of the mechanism. Simply, by driving the inputs at the same angular velocity (same magnitude and direction) will cause the output carrier (gray) to rotate about the same axis as the inputs, shown as the red dashed line in Figure 4.2. Driving the inputs with opposing angular velocities (same magnitude, opposite direction) will drive the output (blue) about its own axis at that angular velocity, shown as the blue dashed line in Figure 4.2. By varying the relative angular velocities of the inputs, combinational outputs are achieved. The inverse kinematic relationships for this mechanism are then described by

$$\theta_0 = (G)\rho + \phi \quad (1)$$

$$\theta_1 = (G)\rho - \phi \quad (2)$$

Where ρ and ϕ are the pitch and yaw angular velocities, θ_0 and θ_1 are the input shaft velocities, and G is the ratio between the input wheel diameter(s) and the output wheel diameter. As shown, the kinematic relationships are simple and linear, making the mechanism computationally efficient to control.

⊣ Head Mechanism Transmission

Our implementation of the differential mechanism uses a transmission system of pulleys and cables, which we selected after reviewing the different transmissions that can be used. The three transmission methods we identified are the chosen pulley-cable system, friction wheels [48], as well as bevel gears, which is the most commonly recognized differential implementation.

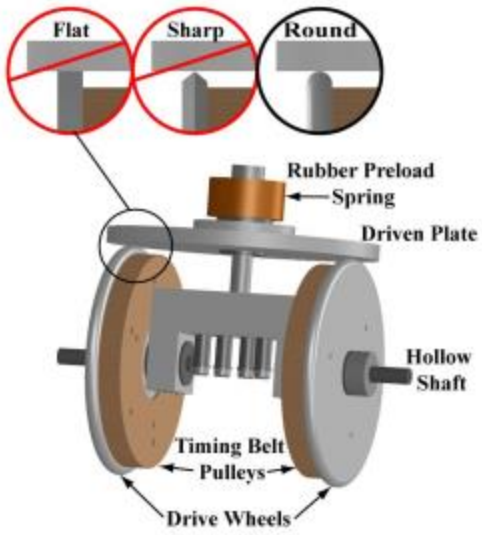


Figure 4.3 Diagram of Friction Wheel Differential

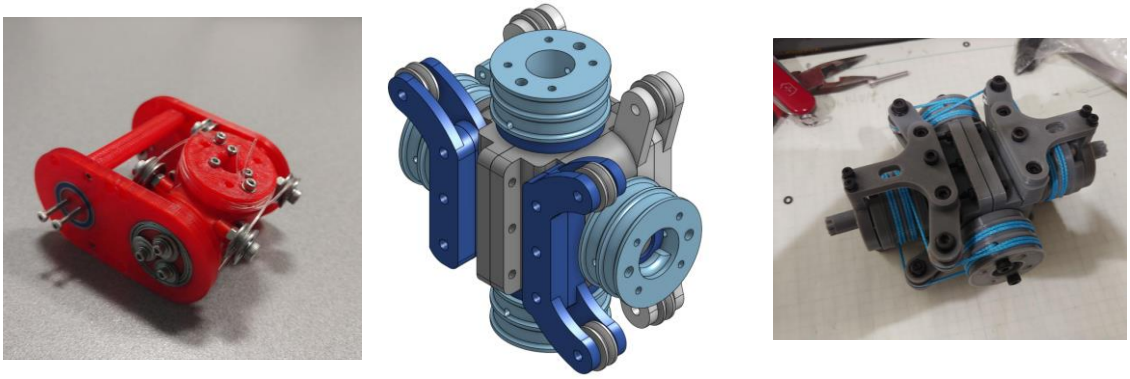
The friction wheel-based differential (Figure 4.3) [48] uses drive wheels forced against an output plate to drive the mechanism. The maximum loading capabilities of a friction wheel-based system is dependent on the preload force between the input and output wheels to generate adequate friction. That force required is quite high [48], which means that materials with high coefficients of friction would be needed. There are many materials that we could use to achieve this, however, the friction interface presents a suboptimal point of failure. If the friction surfaces become dirty, or as the materials wear over time, the frictional force will decrease. For this reason, the safety factor would need to be increased to account for this, which further increases the initial preload requirements. While it is the most mechanically simple of the three options, the reasons outlined make it a less than ideal choice.

Bevel gears provide a rigid transmission between the inputs and outputs and are only slightly more complex than friction wheels. However, backlash and cost become problematic. Extra design work would need to be done to mitigate backlash such as implementing a preload method, and/or higher tolerance parts would be needed. The cost of using bevel gears would not be small as higher precision parts would be needed, and at relatively large sizes.

The last option is to use what we ultimately decided on: pulleys and cables. There are two different ways in which pulleys and cables can be implemented, which we will call the Continuous Cable and Anchored Cable Segments variants. The Continuous Cable variant is articulated by a single, continuous cable routed around each of the pulleys of the mechanism. The Anchored Cable Segments variant is conceptually the same as the Continuous Cable variant, however the cable is instead broken into four parts, each one anchored on both ends.

For our purposes, we decided that the Multiple Cable variant was the best option. The Single Cable variant is mechanically simpler; however, it suffers from similar issues as the friction wheel transmission discussed previously. Additionally, it requires a single, closed loop cable, which would be difficult to source.

⊣ Head Mechanism Cables and Routing



(a) First Prototype (Standard Winding) (b) CAD model of "switchback" version (c) Prototype "Switchback" version

Figure 4.4 Differential Mechanism Cable Designs and Routing

In our design, we are using a UHMWPE fiber rope, which is routed using what we are calling a "switchback" configuration. Figure 4.4a shows the initial winding option, which does not employ this "switchback" design, and Figure 4.4b and Figure 4.4c show the CAD model of the current mechanism, which utilizes our "switchback" routing, along with the prototype version used for evaluation. The prototype allowed us to test different cable materials.

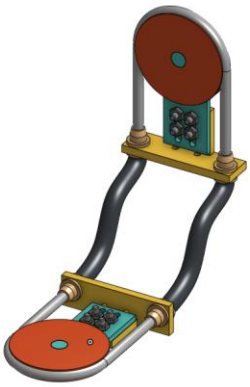
The cable material needed to have a high strength, low elasticity, as well as a relatively small minimum bend radius (i.e. low stiffness). Too large of a bend radius

would increase the pretension required to keep the cable taut and would also introduce additional torque requirements to actuate the mechanism as more force would be required to bend the cables around the pulleys. When evaluating our prototypes for the mechanism, we also found that the cable needs to have low friction with the pulleys for smooth operation. We found that UHMWPE (Ultra-high-molecular-weight polyethylene) fiber rope [50], [51] was a suitable option for us due to its low cost, high strength, low stiffness, and low stretch. Typical UHMWPE fiber rope has a breaking strength of Additionally, UHMWPE has low friction and high abrasion resistance, so minimal or no lubrication is required, reducing maintenance [50], [51], [52], [53]. We had also considered steel wire rope, however, we found that the minimum bend radius was too large for our purposes. More on this in section 5.1 |.

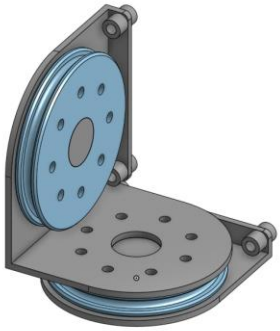
⊣ Additional Head Mechanisms Considered

In addition to the cable driven differential, we also considered some other mechanisms before we could select it as an effective solution. Our investigation focused on four mechanisms: a Bowden cable transmission-based gimbal, Omni-Wrist III [54], a U-Joint Supported platform, and the ultimately chosen cable driven differential. The U-Joint Supported 2DoF platform was originally designed in the previous project term; however, design modifications were made this term based on previous findings. Each of the three mechanisms not chosen were initially analyzed for some unique property they had that would benefit the system, however they were dismissed once we determined they were not the most effective solution. What follows is a discussion of each mechanism, along with our rationale for their initial consideration as well as their ultimate dismissal.

Bowden Cable Transmission Gimbal



(a) CAD model of a pulley pair



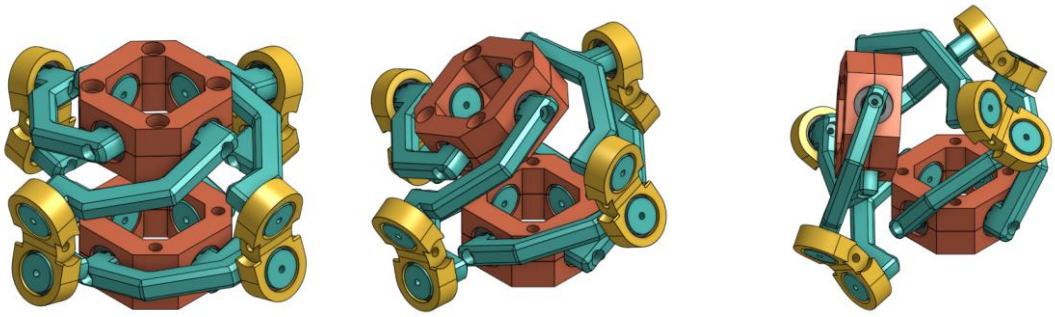
(b) CAD model of a gimbal implementation

Figure 4.5 Our Bowden Cable Transmission CAD Model

This mechanism, or rather transmission (Figure 4.5b) that we have conceptualized a gimbal mechanism implementation (Figure 4.5b), uses a continuous cable (gray, Figure 4.5a) fed through a Bowden tube (black, Figure 4.5a) to couple the rotation of two pulleys. Unlike a traditional pulley transmission, the Bowden tube removes the need to tension the cable by fixing the distance between the pulleys. The pulley can be moved around freely, and motion still transmitted from the input to the output. This property was worth investigating as it could potentially alleviate some design difficulty with transmission routing for other mechanisms. It would also allow us to combine multiple degrees of freedom into a relatively compact space by 'stacking' the output pulleys, as shown in Figure 4.5b.

Despite the potential benefits, the Bowden cable transmission proved to have enough difficulties associated with it that it would be problematic to implement. The first was that the cable would have considerable friction on it by the Bowden tube. This resulting decrease in efficiency could be mitigated by selecting a cable and Bowden tube with correct relative diametric tolerances, however it would be considerably difficult to do so [55]. Additionally, we identified that designing a method for tensioning the cable would be difficult, but not impossible. With more time available for testing, the Bowden cable transmission might still be a viable option, however development time is an important criterion for us.

Omni-Wrist III

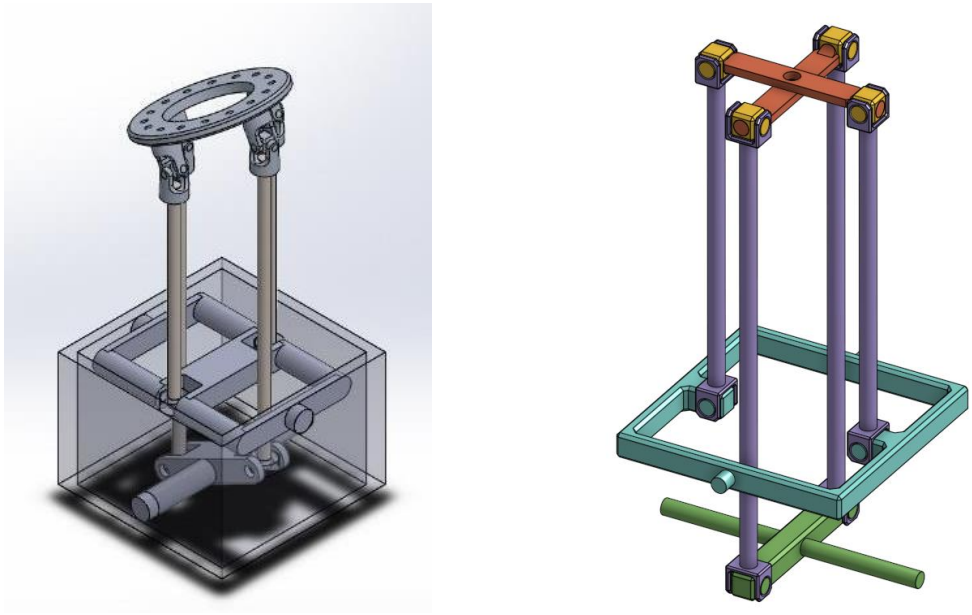


(a) Neutral Position (b) Partial Articulation, Single Axis (c) Full Articulation, Both Axis

Figure 4.6 Our Omni-Wrist III CAD Model

The Omni-Wrist III (Figure 4.6), developed by Ross-Hime Designs, Inc, is an elegant 2DoF mechanism, however it is not very mechanically simple, and it also has a complex set of kinematic equations [56],[57],[58]. What led us to initially consider it was its large range of motion, offering up to, or even greater than 180 degrees of articulation in both rotational axes. While our requirements did not necessitate this large range of motion, it meant that there would be considerable possible range of motion left even after making the mechanism more compact and mechanically robust. What ultimately drove us to dismiss the mechanism was the difficulty of controlling the mechanism. Ross-Hime Designs has not released a complete set of kinematic control parameters, and the research that we found on analysis of the mechanism described a complex set of equations that were very computationally expensive [58].

U-Joint Supported 2DOF Platform



(a) (b)

Figure 4.7 U-Joint Platform CAD models

This parallel mechanism operates similarly to the ubiquitous Stewart platform; however, the design has been modified such that it is constrained to only two degrees of freedom by utilizing U-Joints to link the control rods with the effector platform. The U-Joints prevent rotation along any axis other than those of the control actuators. Two variations of this mechanism were considered (Figure 4.7). We discovered that the first version of the mechanism (Figure 4.7a) was over constrained. The second version (Figure 4.7b) incorporates an additional rotational joint (the interface between the two red members). The reason for initially considering this mechanism was because the control rods (purple) could serve as both the support structure for the neck in addition to transmitting motion from the base of the neck to the head.

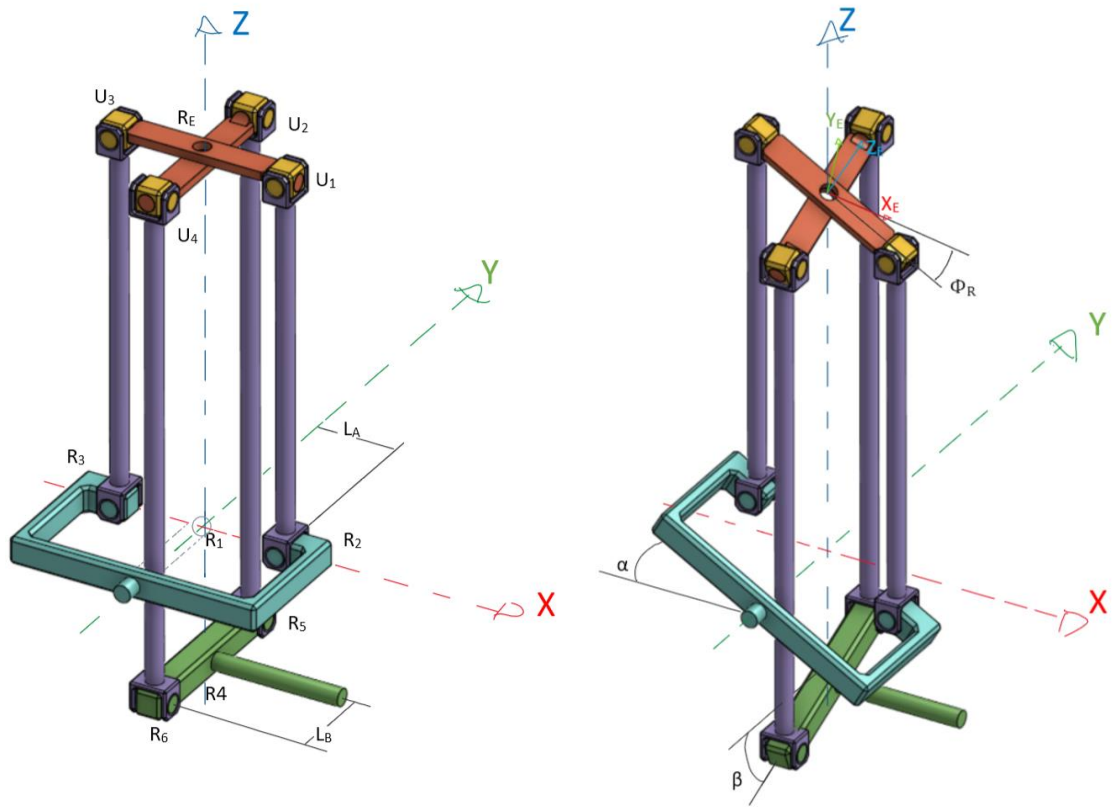


Figure 4.8 U-Joint Platform Nomenclature

To determine the practicality of this mechanism, we first started by examining its geometric properties. Importantly, we needed to understand how the magnitude of the induced rotation (Figure 4.8, red) is affected by different parameters of the mechanism. Additionally, this analysis gave us a function that we could use to compensate for the rotation with a serially attached roll stage, mounted on the effector frame.

Our analysis began by introducing the induced angle of rotation $\underline{\phi_r}$ about the roll axis, $\underline{Z_E}$, of the effector platform,

$$\phi_r(L_A, L_B, \alpha, \beta) \tag{3}$$

where L_A and L_B are the control link lengths shown in Fig. 1a, and α and β are the control angles about the y and x axis, respectively, as shown in Figure 4.8b. We define $\vec{L_A}$ as the position vector from R_1 to R_2, and $\vec{L_B}$ as the position vector from R_3 to R_4. Note that R_1 is projected to the xz plane, and R_3 is projected to the yz plane.

$$\vec{L_A} = \begin{bmatrix} L_A\cos(\alpha) \\ 0 \\ L_A\sin(\alpha) \end{bmatrix} \vec{L_B} = \begin{bmatrix} 0 \\ L_B\cos(\beta) \\ L_B\sin(\beta) \end{bmatrix} \tag{4}$$

To find our angle, ϕ_r, we will simply use the law of cosines. First, we find the distance, S_E, between U_1 and U_2, which is equivalent to the distance between position vectors $\vec{L_A}$ and $\vec{L_B}$.

$$S_E = \sqrt{(L_A\cos(\alpha))^2 + (-L_B\cos(\beta))^2 + (L_A\sin(\alpha) - L_B\sin(\beta))^2}$$

We can now apply the law of cosines to find the total internal angle between the platform, θ_t

$$\theta_t = \cos^{-1}\left(\frac{S_E^2 - L_A^2 - L_B^2}{-2L_AL_B}\right)$$

We are concerned with the rotation of the platform relative to the platform x-axis, rather than the internal angle, so we say

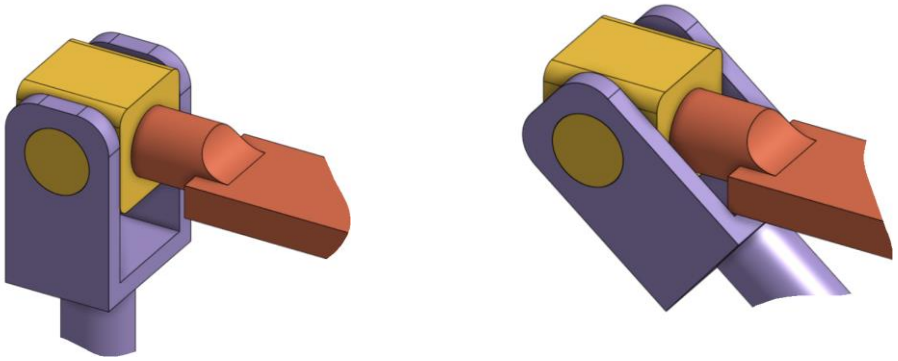
$$\phi_r = \frac{\pi}{2} - \theta_t$$

Finally, composing the above work yields

$$\phi_r = \cos^{-1}\left(\frac{L_A^2\cos^2(\alpha) + L_B^s\cos(\beta) - L_A^2 - L_B^2}{2L_AL_B} + \frac{L_A\sin(\alpha) - L_B\sin(\beta)}{2L_AL_B}\right) - \frac{pi}{2}$$

Simplifying the above shows that the angle of rotation about the roll axis does not, in fact, depend on L_A or L_B. Thus, our final equation for ϕ_r is

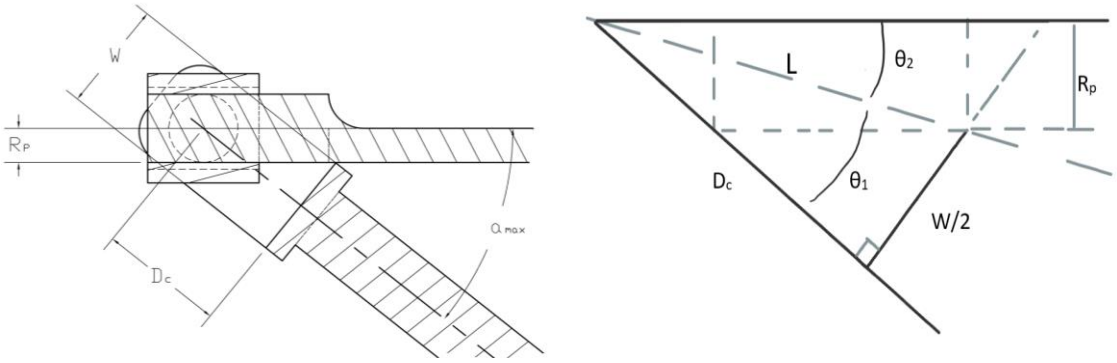
$$\phi_r(\alpha,\beta) = \frac{\pi}{2} - \cos^{-1}(\sin(\alpha)\sin(\beta)) \tag{5}$$



(a) (b)

Figure 4.9 Zoomed View of U-Joint Platform Joint. Shown: yoke (purple), spider (yellow), platform pin (red)

Seeing that the induced roll is reasonably correctable, we continued our evaluation by examining what geometric elements of the components would affect its motion range. We first looked at the effects of the U-Joint geometry. Zoomed views of the components in question are shown in Figure 4.9; (b) illustrates the configuration in question. The Platform will reach its angular articulation limit when the yoke contacts the platform. The pin is assumed to be long enough, so the yoke does not contact the platform. In a real implementation, the pin need only be long enough for the spider, and the remaining required pin length may be of different geometry.



(a) (b)

Figure 4.10 U-Joint Platform Cross-Section Geometry

The maximum articulation angle, denoted a_{max}, will depend on the radius, R_p, of the platform pin, the base width, W, of the yoke, and the base clearance, D_c of the yoke. To start, we divide a_{max} by a line, L, connecting the rotational center of the yoke and the point of contact. We say that

$$a_{max} = \theta_1 + \theta_2$$

From the Pythagorean theorem, we have

$$L = \sqrt{D_c^2 + \frac{W}{2}^2}$$

Using some trigonometry, we find

$$\theta_1 = \tan^{-1}\left(\frac{W}{2D_c}\right), \theta_2 = \sin^{-1}\left(\frac{R_p}{L}\right)$$

Finally, we arrive at the desired expression

$$a_{max} = \theta_1 + \theta_2 = \tan^{-1}\left(\frac{W}{2D_c}\right) + \theta_2 = \sin^{-1}\left(\frac{R_p}{\sqrt{D_c^2 + \frac{W^2}{4}}}\right) \qquad (6)$$

4.2 | Electrical Systems Design

It would not be possible for the robot to interact with users and its environment without a way to process input. This includes audio, physical, and visual stimuli. To achieve this, the robot needs a set of sensors and processing components that enable it to sense the world around it and act accordingly. While careful design can reduce the potential for failure, it is still possible for external issues or unexpected input to cause a fault. This understanding is what drove our component decisions and designs.

We started the process for the design of the electrical systems by determining sensor and processing requirements. This was done in coordination with the mechanical designs as they influenced the location of the touch sensors, camera and microphones, as well as the camera field of view. We also worked with the designs of the mechanical systems to determine actuation requirements. After researching and selecting suitable component candidates that could meet our specifications, we determined the system power requirements, and set a power budget.

We then used all this information to draw a comprehensive electrical system schematic, which included the necessary circuitry for a fully functioning system. Namely, we needed to design the power distribution and protection circuits and add in components to maintain signal integrity. Our schematic also included necessary auxiliary components as specified by various component manufacturers that ensure proper component settings and function. Schematic outlining was first done on using a web-based diagraming software provided by Digi-Key, our primary source for electrical components and hardware. Though the software does not offer all the features offered by dedicated circuit design software, its small learning curve and integration with their parts catalogue made it a time saving tool.

Despite the convenience, Digi-Key's diagraming software was ultimately not suitable for our final designs. Upon recommendation from students in the ECE department, we considered Autodesk Eagle and KiCad. While both offer similar capabilities, Autodesk Engle proved to be an easier solution to learn. As an industry

standard, it is likely Eagle will also be more familiar to future teams. More information on that can be found later in the results section.

4.2.1 | Processors

This robot processor must be capable of collecting and processing input from sensors and peripherals, and then using it to compute and rapidly execute dynamic actions. More importantly, the robot must be able to do this reliably and consistently to maintain its animated character. Skips or lags due to lack of adequate CPU time might incorrectly indicate malfunction to the user, breaking the 'character,' or possibly causing more adverse reactions resulting in harm. While careful design can reduce the potential for failure, it is still possible for external issues or unexpected input to cause a fault. This understanding is what drove our component decisions and designs.

As a robot designed for a dynamic production environment, we recognized that our processor needed to be fault tolerant and easy to reset. To address this, we outlined several requirements. First, the processor needed a boot-up time of no longer than 30 seconds. This requirement was deemed a reasonable period the average person would wait for a computer to restart. The robot must also have a position control bandwidth of 7.5Hz and a vision system bandwidth of 15Hz. The bandwidth of a system is the response time of that system. The position control bandwidth requirement was determined based on the value used for a similar multi-DOF robot [18]. The vision system bandwidth was determined as a reasonable bandwidth to handle the larger amount of information images require as compared to positional values. Finally, it is preferred that the processor does not utilize an operating system. With an operating system comes boot time, so the robot would need to take time to restart. An operating system also runs the risk of crashing, forcing the robot to restart and waste precious time, which would be especially problematic during a public event or presentation. However, a processor without an operating system would be able to restart nearly instantly should something go wrong.

As a starting point, we specified that all the onboard processing was to be handled by a single real-time MCU, in this case an ESP32. The ESP32 is a low-cost, yet still very capable MCU from Espressif Systems [19]. While it is most popularly used for its full onboard hardware networking stack, its fast (base 160MHz, 240MHz configurable) dual core architecture makes it an attractive option for the tight timing requirements of our robot. It additionally has many of the hardware peripherals that make robot development easier, such as multiple serial interfaces as well as hardware ADCs, DACs, and a host of signal processing functions built right onto the silicon. Development for the ESP32 is also made more accessible by its integration of FreeRTOS, an open source real time operating system [20]. This alleviates the often-cumbersome job of designing task schedulers and ISRs for low level, timing critical applications without adding the unpredictable overhead of a full OS. For general robot applications, the ESP32 is capable of handling sensors and motor control, both autonomously and tele-operated. The ESP32 MCU does have its limits, however, and it lacks the processing power to handle complex AI calculations. While the ESP32 performs well under reasonably deterministic conditions, there are limitations to its capabilities.

To address the weaknesses of the ESP32, an external processor such as a laptop or Raspberry Pi, that would wirelessly connect to the ESP32 was considered. However, as the design evolved, it was decided that a Raspberry Pi would directly connect to the ESP32 boards inside the robot. While Raspberry Pi utilizes an operating system, which we initially sought to avoid, it was ultimately decided that the unpredictable nature could be dealt with, as it would perform only bulk operations. Having an OS, in this case a Linux distribution, means creating user interfaces (in our case simply over the command line), is much faster. Although not as powerful as a laptop or PC, the Raspberry Pi is still a versatile computer despite its small size. While the ESP32 boards would handle the I/O of the sensors and motors, the Raspberry Pi would handle the AI of the robot.

Espressif Systems offers the ESP32 as an integrated surface mount package, or as part of a number of development board options, each tailored for different applications. Two ESP32 development boards were considered—the

DevKitC and the WROVER-VB. Unlike the other development boards offered by Espressif, the DevKitC is a general-purpose development board designed to be easy to use and integrate with a breadboard. Its low cost of $10-$18 made it an affordable option. Meanwhile, the WROVER-VB is a more advanced version of the DevKitC. It includes features such as PSRAM, support for LCD and microSD, and a multi-protocol USB bridge. However, these extra onboard devices also mean the WROVER-VB is more expensive than the DevKitC at $40. However, these features are not necessary for our purposes, making the standard DevKitC the most sensible option.

Similarly, two Raspberry Pi models were considered, the Raspberry Pi 3 Model B+, or the Raspberry Pi 4 Model B. The two processors have similar specifications, with the Raspberry Pi 4 having a slightly better processor and memory options, running at 1.5GHz with RAM of up to 4GB [21], compared to the Raspberry Pi 3's 1.4 GHz with only a 1 GB RAM option [23]. However, a common complaint with the Raspberry Pi 4 is that it experiences hardware problems. Among other issues, a particularly concerning issue with the Raspberry Pi 4 is that it gets hotter and more prone to temperature throttling than the Raspberry Pi 3 [23]. Meanwhile, the Raspberry Pi 3 is well documented and still supported and does not have the same hardware issues. This difference in reliability was the deciding factor in using the Raspberry Pi 3 Model B+.

The design we used for the processor became a combination of the Raspberry Pi and ESP32 communicating through SPI protocol. Initially, the Raspberry Pi would act as the SPI slave and handle the AI and visual sensor of the robot, while an ESP32 would act as the SPI master, taking information from the motors and sensor and sending them to the Raspberry Pi to be processed. Commands for how to move the motors would be sent back from the Raspberry Pi, and the ESP32 would act accordingly. This would allow the robot to have a safety net if the Raspberry Pi crashed, allowing the robot to continue operating should the Raspberry Pi fail as the ESP32 could be designed to not rely on the Raspberry Pi.

The design changed along the way, as we soon realized the ESP32 would not be capable of handling all of the motors and sensors, due to lacking the physical pins

to support both the Raspberry Pi and the sensors and the amount of information that would need to be transferred. We revised the design in two ways. The first was to make use of two ESP32's - this allowed for more connections to sensors, and each ESP32 could be given a specific role. One ESP32 would collect data from the sensors. The other ESP32 would handle the motors. The second revision was to make the Raspberry Pi a SPI master and the ESP32s SPI slaves. Although this contradicts the preferred requirement to not rely on an operating system, the better performance and physical pins of the Raspberry Pi made it much easier to work with. With the split roles of the ESP32s, we were concerned that making one ESP32 a SPI master would overcomplicate the system. With the higher processing power of the Raspberry Pi, it would likely be able to handle the communication much better than a single ESP32 could.

4.2.2 | Imaging and Computer Vision

As a part of interacting with people in front of it, the robot needs visual sensors to see what the world around it is. For this robot, it needs to be able to identify people within its viewing range, and seek out people based on faces, bodies, or hands. For this reason, cameras are necessary to detect the finer details of people. For this reason, vision systems such as ultrasound, infrared, and LIDAR are not viable options.

The camera of the robot has two must-have requirements. The first requirement is that the camera should have about 120° of horizontal FOV and 40° of vertical FOV. Although an actual goat's FOV is significantly higher at 320°—340°, it is unlikely that all those degrees are necessary for the operation of the robot. Most of the interaction with the robot will occur almost directly in front of the robot probably within the front 90° as this is where much of the interaction between humans happens. Any excess beyond that would mostly be used to look for people to interact with. For instance, if the robot is seeking someone to interact with and someone is spotted in the FOV, the robot will turn to that person. Therefore, any more degrees beyond 90° are rarely being used, and will require more processing power, and camera view blending. Given available models, 120° was determined to

be an appropriate number that allows the robot to still notice movement outside the area directly in front of it but also balanced with the necessary processing power for one or two cameras. However, 120° is a large FOV for standard cameras. Therefore, two cameras, each with about 60° of horizontal FOV, would also be an acceptable solution as together they would provide the desired FOV.

The second requirement is that the camera needs to have a minimum frame rate of 30 FPS. This was decided to be a reasonable requirement, as 30 FPS still generates images where objects can still be tracked. Any lower may make the footage too choppy to track a single object without mixing up objects, and no higher bound was set as the FPS of a camera can be lowered as necessary. Finally, a preferred capability of the camera is that allowing the robot was capable of being able to see the ground at a minimum of 1m in front of its chest. Despite these requirements, only one camera will be used in this iteration of the project. This is done for several reasons. First, 60° of horizontal FOV is still a reasonable FOV for a robot. Second, reducing the number of cameras will save time from trying to combine and analyze footage from two cameras. The purpose of this robot is to be a proof-of-concept for future projects to build on. It would be a waste to invest too much time and effort into cameras that can be potentially replaced by a better solution. It is more important that the robot show its ability to respond to visual stimuli, rather than have a wide FOV; while having a wider FOV would be beneficial for future robots, the time and effort spent on joining the images using the camera's relative positions would be better spent on the proof of concept of actually using object detection on the resulting image.

Figure 4.11 ESP-EYE Camera Board

The first camera considered was the ESP-EYE (Figure 4.11). The ESP-EYE is an ESP32 development board designed by Espressif. Unlike the DevKitC, The ESP-EYE is designed for image recognition and Artificial Intelligence of Things applications [24]. The built-in camera is a 2-megapixel OV2640 camera. It has a horizontal FOV of 56°, a vertical FOV of 40°, and a framerate of 30 FPS [25]**.** However, the ESP-EYE's only form of connection is a USB port that handles information and power transmission for the system. To utilize the ESP-EYE, it would need to connect to the Raspberry Pi via its USB port, which contradicts one of the requirements. Additionally, the ESP-EYE lacks detailed documentation, which may make it difficult to work with.

Figure 4.12 Pixy2CMUcam5 Image Sensor

The next camera is the Charmed Labs Pixy 2 CMUcam5 Image Sensor (Pixy 2) (Figure 4.12) [26]. The Pixy 2 supports multiple interface options and is therefore capable of connecting to several types of controllers. Recently it has added support for Arduino and Raspberry Pi platforms. Like the ESP-EYE, the Pixy 2 is also capable of image recognition, and with a framerate of 60 frames per second (FPS), the Pixy 2 can take smooth video. The lens of the Pixy 2 has a FOV of 60° horizontally, and 40°

vertically. Being an image sensor, the Pixy 2 already has built-in image recognition features. By pressing the button, the Pixy 2 can be taught what images it should recognize. While this feature would be helpful, it defies the point of designing the software to handle image recognition on its own. However, the Pixy 2 is also significantly more expensive than the other options at $60 each.

Figure 4.13 Arducam with M12 lens module

The third camera is the Arducam with the M12 lens module (Figure 4.13) [27]. The Arducam is designed for use exclusively with the Raspberry Pi, as it can be plugged directly into the camera port of the Raspberry Pi board. It can record at multiple resolutions at different frame rates. It has 30 fps at a resolution of 1080p, 60 fps at 720p, and 90fps at 480p. With the M12 lens, the Arducam has a horizontal FOV of 56°, making it slightly short of the acceptable FOV specification, but not by a large amount. As Raspberry Pi's only have one camera port, it will be impossible to use two of these cameras. However, the Arducam has a low price of $19, which makes it a good option for testing and working with a camera feed.

The camera we decided to use was the Arducam with the M12 Lens. As stated before, the current project only requires one camera to show that the robot can react to visual stimuli. The Arducam connects to the Raspberry Pi via the Raspberry Pi's camera port, making it easy to integrate. Although the Pixy 2 would be a more ideal solution given its capabilities as an image sensor, its high price point makes it a large investment from a budgetary standpoint. Additionally, having image recognition capabilities built-in defeats the purpose of the project. The built-in firmware and applications that come with the Pixy 2 means more effort will have to

be spent on integrating, and potentially fighting, the existing systems of the image sensor. The Arducam has similar performance specifications for a much smaller price.

4.2.3 | Microphones and Audio

Modern interactive robots are typically expected to be responsive to various audio cues, be it in the form of verbal communication, or simply making noises at it, like clapping. Not only does the robot need to be capable of detecting the presence of these sounds, but it must also be capable of determining the source of origin. This implies a set of strategically placed microphones around the robot. For this iteration of the robot, it was decided that only the detection of sound presence would be necessary, leaving room for future projects to expand the robot's capabilities, such as voice recognition. Although voice recognition would be a good feature for the final robot. It is not a focus for this iteration.

For audio detection, a set of three microphones will be used. They will be placed on the front, right, and left of the robot. The three microphones will not only allow the robot to detect sound, but also locate the source of the sound— something necessary for the robot to be able to turn and face the source auditory cues outside of its FOV. The requirement for the sensitivity of the microphone was chosen to be about -50 db. The average voice level at 2 meters away is approximately 54 dB [25]**.** 2 meters was considered a reasonable maximum distance for a person to stand away from the robot while talking to it. The sensitivity of a microphone is the minimum volume that it can detect. Therefore, it was necessary to find a microphone with a sensitivity of at least -50dB, but not too much greater than the said -50 dB, otherwise the microphone would be too sensitive to sound. Microphones with wire leads were prioritized, as they would be the simplest to integrate with the protoboard.

When evaluating microphones, there are 3 types of microphones that are commonly used: noise canceling, omnidirectional, and unidirectional. Noise canceling microphones are designed for sources that are close to the microphone, without detecting extraneous noise. They are generally used for applications such as

headset or podium microphones. Omnidirectional microphones can detect noise from all directions, which is useful when the direction of the sound does not matter. However, using multiple omnidirectional microphones runs the risk of overlap happening. This happens when multiple microphones pick up the same sound. Since the microphones we are using in particular will not be able to detect the direction the sound is coming from, the overlap will likely confuse the robot. For this application, unidirectional microphones are the preferred solution. Unidirectional microphones only detect noise in the direction they are pointed.

Figure 4.14 AUM-5047L-3-LW100-R Microphone

The first microphone is the AUM-5047L-3-LW100-R (AUM-5041) (Figure 4.14) [26]. This unidirectional microphone is rated for 1.5V with a range of 1.5V to 10V and has a sensitivity of -47db ±4 dB [27]. It has wire leads, making it easy to connect to a proto-board or bread board.

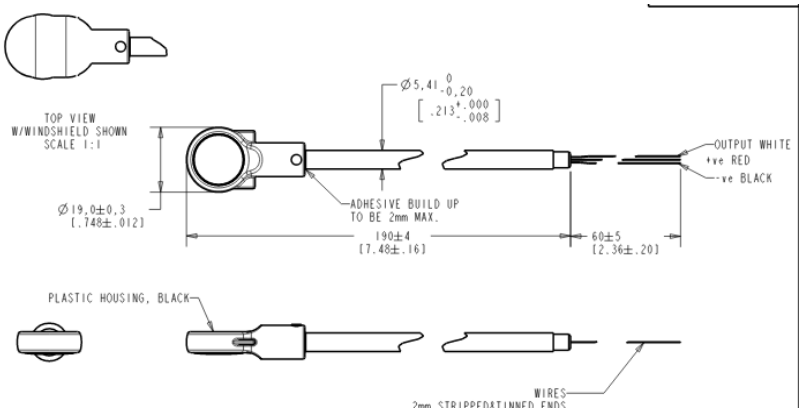


Figure 4.15 FB-EM-30346-000 Microphone

The next microphone is the FB-EM-30346-000 (FB-EM) (Figure 4.15) [27]. This microphone is rated for 1.3V with a range of 1.3V to 10V and has a sensitivity of -48db ±3 dB at 74db SPL [27]. Unlike the AUM-5041, this microphone is an omnidirectional boom microphone. However, being a boom microphone, the FB-EM

is much more expensive than an electret condenser, making it a difficult choice to go with when considering the budget.

The last two microphones are the POM-2246L-C33-LW100-R (POM-2246L) and the CMC-3015-44L100 (CMC-3015), which are nearly identical [28]. Both are omnidirectional electret condenser microphones with similar voltage ratings and sensitivity levels. Both microphones are rated for 1.3V with a range of 2V to 10V. However, the POM-2246L has a sensitivity of -46db ±3 dB [28], while the CMC-3015 has a sensitivity of -44db ±3 dB [32].

We decided to use the AUM-5041 as it is the only microphone that is both unidirectional and is close to the desired sensitivity specification. Its low price point of $2 makes it a cheap investment. As the other microphones were omnidirectional microphones, they ran the risk of causing overlap that could confuse the robot. The microphones will be integrated into the robot through its circuit board. The input (red) line both receives power and sends signals from the microphone. That line will be connected to the ESP32 which will receive the signals from the microphone.

4.2.4 | Touch Sensors

The last need for the sensors is the ability to sense touch. As an interactive robot, the robot needs to be able to react to being touched. People will inevitably want to touch the robot, especially if it was designed to interact with people. By behaving differently according to where it was touched in combination with its other behaviors, the robot can sell the image of a character.

For the capacitive touch sensors, it was determined that they must be able to detect a touch through at least 3 mm of material and that at least 6 sensors are necessary for adequate coverage. These 6 sensors would cover much of the neck and head areas which is where most of the physical interaction is expected to happen; these 6 sensors would cover the left neck, right, neck, left head, right, head, top head, and torso of the robot. The reason for the 3 mm of material is that a capacitive touch sensor will probably be spread across a foil skin behind an exterior layer of material. Although 3 mm is larger than necessary since the exterior layer is unlikely

to be that thick, it is a safe limit since the exterior layer has yet to be finalized. However, there are currently no plans for the cover to be created for this project. Therefore, it will only be necessary to have nodes on the robot that simulate the touch points of the robot. That being the case, whether the touch sensors can detect touch through the cover is not important for this project.

Although it is technically possible to operate with only one touch sensor, the robot would exhibit limited behavior. Animals react to touch differently depending on where they are touched. It was determined that at least 6 sensors would give the robot enough different types of motion. Two would be placed on the body of the robot on its left and right flanks, which would allow the robot to turn its head and neck to that side and react. Two more sensors would be placed on the left and right of the neck which would limit the speed and range of motion of the neck when being touched. Finally, two would be placed at the top of the head, one at the back of the head, and one for the top of the face and its horns. These would probably be the most used and allow the robot to nuzzle hands.

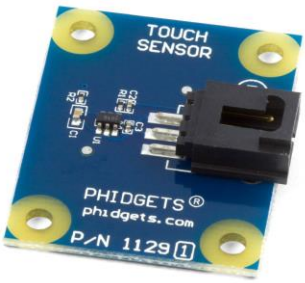


Figure 4.16 Phidgets Touch Sensor

The first capacitive touch sensor is the Phidgets Touch Sensor 1129 (1129) (Figure 4.16) [33]**.** This capacitive touch sensor is capable of detecting a touch through plastic, glass, or paper; up to ½" thick. Although the sensor is small, additional connections can be soldered onto the sensor to increase its range. The sensor is 3.3V compatible, however it also requires a Hub Phidget that connects to a computer via USB [32]**.** This makes it unusable with the ESP32, as it does not have USB hardware.

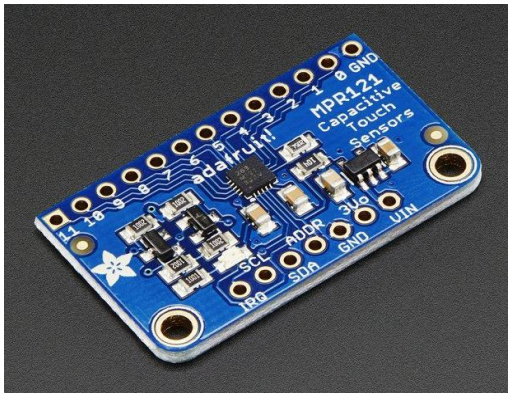


Figure 4.17 Adafruit 12-key Capacitive Touch Sensor

The other capacitive touch sensor considered was the Adafruit Multi-Key Capacitive Touch Sensor Breakout Board (Figure 4.17) [35]. This capacitive touch sensor comes in a 5, 8, and 12-key configuration. Each key is a touch sensor that can be extended with additional wiring. When a key is touched, the board will output a response that the key is being touched. This board allows for a much wider range with only a single board. However, it only works for mediums that are electrically conductive, which limits the materials it can be used with [35].

We decided to use the Adafruit 12-Key Capacitive Touch Sensor Breakout Board, which means only one sensor will be necessary. Additionally, the expandability of the board allows for greater coverage of the robot. Because this project will not address the cover of the robot, there is no need to find sensors that will be compatible for a non-existent cover. When comparing the Adafruit touch sensor to the Phidgets touch sensor, the range and communication capabilities of the Adafruit sensor are far superior. The 12 keys of the Adafruit sensor allow for a greater variety of ranges, while not needing to rely on a USB hub makes the Adafruit sensor much more reliable compared to the Phidgets sensor.

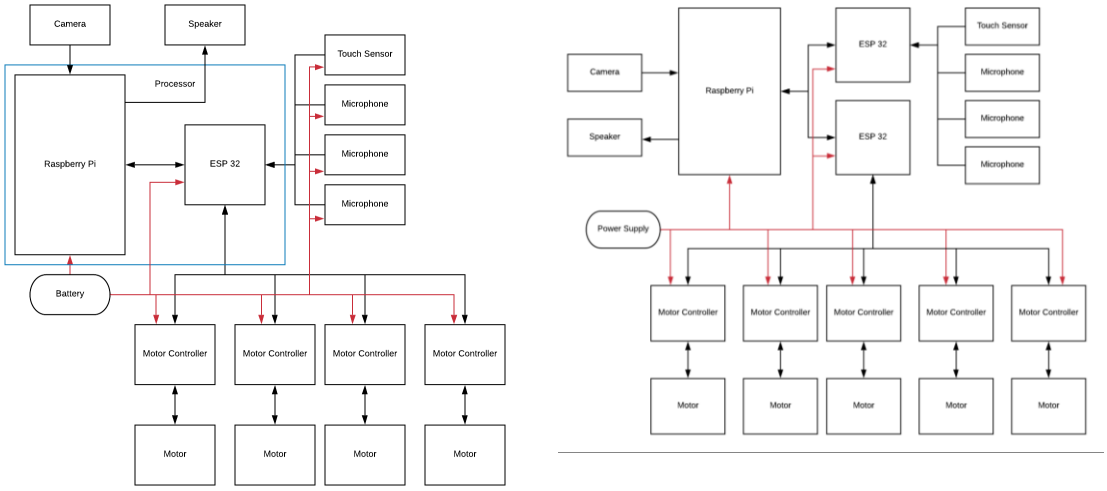
ESP32 boards also can turn most of their GIPO ports into capacitive touch sensors. The ESP32 has more than enough ports to cover our needed 6 touch sensors and upon testing it showed that it could detect a signal through a foil, this sensor also had the advantage of not needing extra communication like with the Adafruit multi key sensor which would delay the system. Touch thresholds could also be set on the ESP that would trigger an interrupt which might be advantageous

for software design. Compared to the previous two options using the ESPs built-in capacitive touch sensors has some clear advantages such as shorter delay times, no need for extra communication software and the option of hardware interrupts.

4.2.5 | Motors

The motors we chose to use to actuate the robot were UCONTRO iHSS57-36-20 integrated stepper motors [36]. These motors are compact and are capable of high torques. These motors have an operating voltage range of 24-50VDC, and nominal operating currents of 3A. However, if the motors encounter high stall torques, the motors may pull much higher currents than 3A. Therefore, to prevent the motors from pulling more currents than the power supply can provide, we will consider the motors capable of pulling 6A. However, doing so also increases the power requirements of the robot.

4.2.6 | System Block Diagram



(a) **(b)**

Figure 4.18 Initial and Final Electrical System Diagrams

Although knowing the components is an important part of designing the robot, what really matters is how the components are assembled. For this reason, a

system diagram can be used to highlight the important components of a system and illustrate how they are connected.

The robot was designed with two sections in mind. The first part of the robot is the high-powered motors used to move the head and neck of the robot. The second part is the processors of the robot. In the initial design (Figure 4.18a), the processor consisted of a Raspberry Pi and one ESP32 module. The Raspberry Pi would handle the AI algorithms and camera data. The ESP32 would handle data to and from the microphones, touch sensor, and motors. Additionally, the ESP32 would be able to handle the manipulation of the head and neck motors.

However, upon working with the physical boards, it became apparent that a second ESP32 would be necessary. The additional board would cut the computational load on a single ESP32, while also having enough physical pin connections for all of the sensors and motors. The system diagram was revised to reflect these changes (Figure 4.18b).

4.2.7 | Power Supply

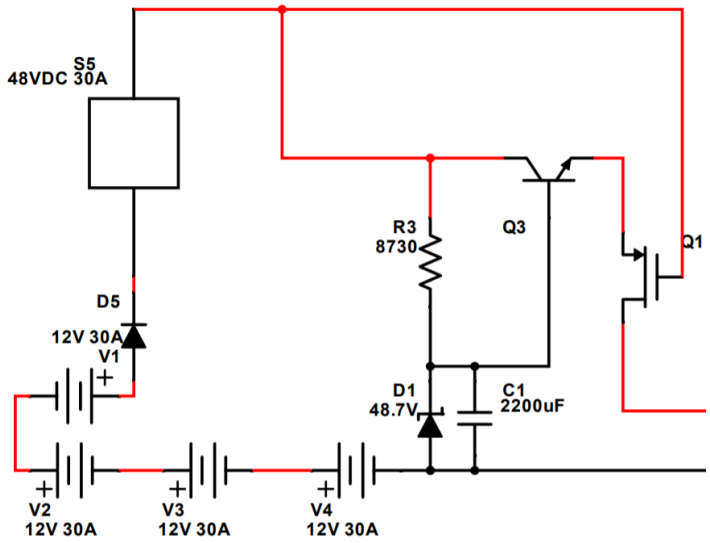


Figure 4.19 Power Supply and Delivery Circuit Diagram

Due to the voltage and current requirements of the motor, approximately 48V and 32A were necessary to run the system. However, single module power supplies capable of providing the required power are generally quite expensive,

costing about $300 at the minimum. However, because voltages in series add together, we were able to substitute a single power supply with four 12V and 30A to generate 48V at 30A (Figure 4.19). Although this is not the 32A we originally desired, this iteration of the robot will not use all five motors like originally planned. Therefore, the power supply can have a lower rated current.

After the power supply is an overcurrent protection device. Should the circuit try to pull more than the rated current, the overcurrent protection will cut the connection. This will prevent an excess of current in the main circuit and protect the circuit from damage if the power supplies malfunction. At the same time, the overcurrent protection also protects the power supply if the circuit tries to draw too much current. Several options were considered for this device. The first was a fuse rated for 30A, should the current exceed 30A, the fuse would blow, cutting the connection to the rest of the circuit. Although this option was viable, fuses need to be replaced once they blow. The other option considered is a circuit breaker. Circuit breakers are generally more expensive than fuses but can be easily reset. Considering that the robot needs to interact with people, it is important for the system to be easy to reset. For this reason, a circuit breaker was the better design decision. For this circuit, we used a 30A circuit breaker rated for 48V.

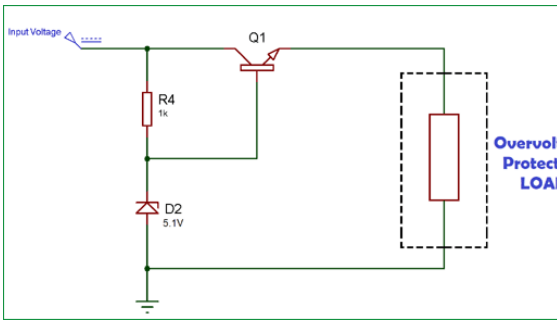


Figure 4.20 Zener Regulator Circuit

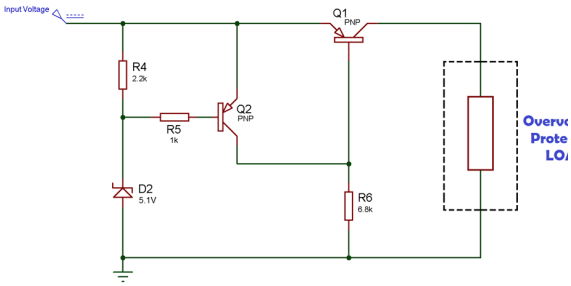


Figure 4.21 Overvoltage Protection Circuit

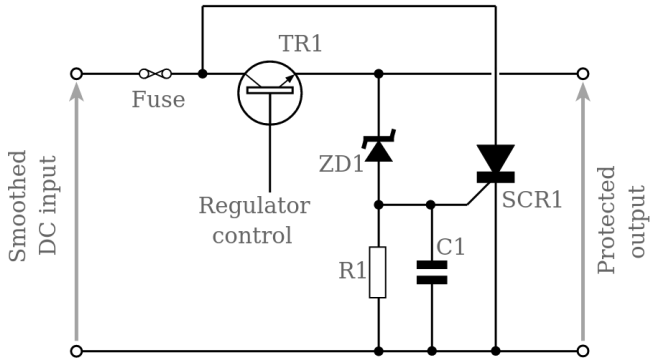


Figure 4.22 SCR Crowbar Circuit

Following the circuit breaker is an overvoltage protection configuration. Although power supplies are reliable, there is still the possibility of the power supply malfunctioning. To protect the circuit from such a malfunction, an overvoltage circuit is used. Some common overvoltage protection circuits are Zener voltage regulator circuits, Zener overvoltage protection circuits, and SCR overvoltage crowbar circuits. Zener voltage regulator circuits (Figure 4.20) use Zener diodes to limit the voltage of a circuit, if the voltage in the circuit overcomes the rated voltage of the diode, the Zener diode will allow current to flow through it. Doing so sends the current to ground, reducing the voltage to the rated level. A Zener voltage regulator will do this without cutting power to the load, allowing everything else to function. In comparison, a Zener overvoltage protection circuit (Figure 4.21) behaves in the same manner as the regulator circuit. However, the overvoltage protection circuit will cut power to the load if an overvoltage condition occurs. SCR Crowbar circuits, in contrast, close a short circuit over the output if an

overvoltage condition is experienced (Figure 4.22). They are often linked to a fuse which will blow if an overvoltage condition occurs.

For this robot, we decided to use a Zener voltage regulator circuit. Although a Zener overvoltage protection circuit is likely safer for users, suddenly cutting power to the circuit runs the risk of damaging the processors from the sudden power loss. The Zener regulator circuit is also preferable over the SCR crowbar circuit because the SCR circuit uses a fuse. As discussed before, a fuse that blows needs to be replaced, which is impractical for this robot. Therefore, compared to these options, the Zener voltage regulator was the most preferable.

4.2.8 | Motor Circuit Diagram

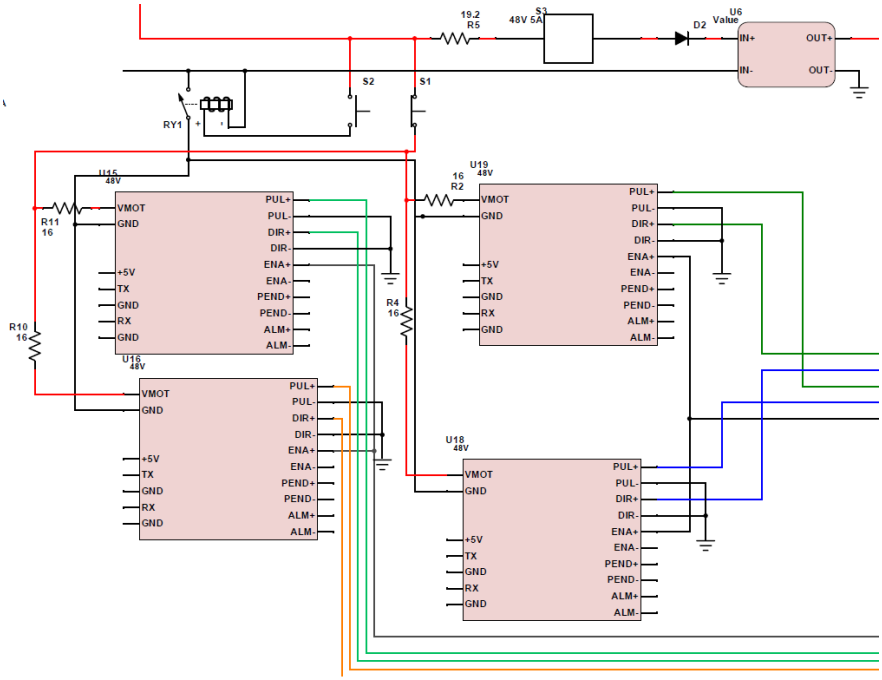


Figure 4.23 Motor Control Circuit Diagram

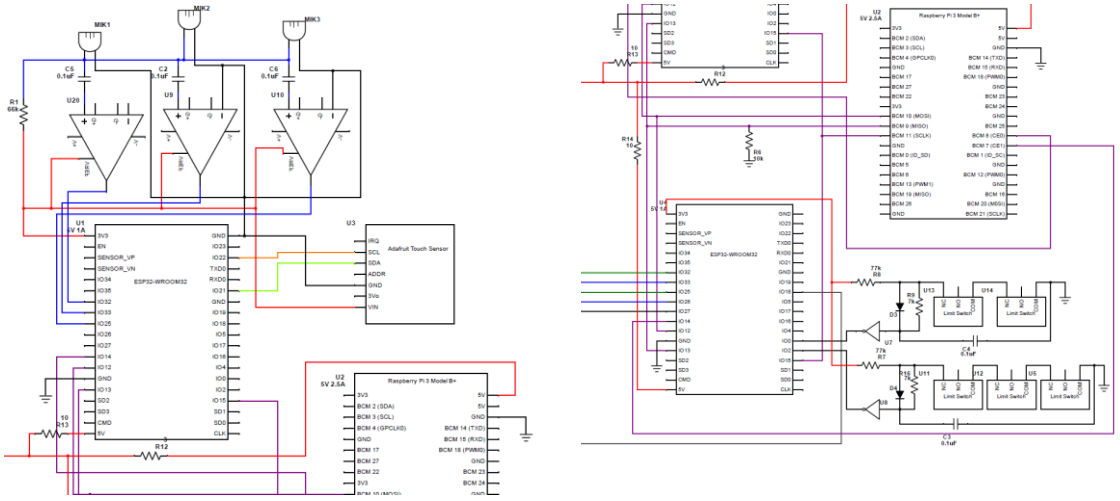
To reduce the complexity of the robot for C-Term, this iteration of the robot's head was not designed to have a roll axis. The motors for the head and neck are UCONTRO iHSS57-36-20 hybrid stepper motors. These motors are rated for operation at 20-50VDC and pull about 3A, handling up to 6A. We wanted to run the motors at 48V, so 16Ω resistors for each motor were used to ensure that the current would be reduced to an acceptable level. The motors have positive and negative pins

for five different functions. The PUL pins are also known as step pins on some stepper motors. Positive signals make the motors take a step. The DIR pins control the direction of the motors. A positive signal to the pins makes the motors rotate in a counterclockwise motion while a negative signal makes the motors rotate in the opposite direction. The ENA pin enables or disables the motors. When left uncontrolled, the motors are enabled. The PEND pins are output pins that are high when the actual position is different from the command position. Finally, the ALM are output pins that are high when a protection feature is activated, such as overvoltage, over-current, or a position following error has occurred. The motors are each connected to an ESP 32 which handles their control (Figure 4.23).

For a physical safety mechanism, an emergency stop (E-Stop) is used between the motors and the power line. E-Stops are a general safety measure that is expected of every robot and can be used in the rare instance that further operation of the robot will result in physical harm. The presence of an E-Stop, while it likely never will be required, is a vital step to ensure that the robot can be switched off immediately and physically. Should the E-Stop be switched, the switch will cut power to the motors and redirect power back to ground. Without power, the motor will be forced to stop immediately which should prevent further harm. Should the robot start malfunctioning, this switch can be flipped to prevent the robot from moving dangerously.

Because the motors and processors are connected to the same power source, there needs to be a way to reduce both the current and voltage going to the processors to prevent them from being damaged. To do so, we used a DC/DC buck converter to reduce the voltage from 48V to 5V. We used a DROK step-down voltage regulator as the converter for its ease of implementation and capability to reduce 65V to 5V. However, the DC/DC converter takes a maximum of 8A, and outputs a current equivalent to the input current. To reduce the current to a more suitable 5A, a resistor connected to a 5A circuit breaker was used. Should the processor circuit try to pull more than it is rated to handle, the circuit breaker will cut the current flow and prevent the processors from being damaged. A diode is also used to keep current flowing in the correct direction.

4.2.9 | Processors and Sensors



(a) (b)

Figure 4.24 Processor Circuit. Sensor ESP32 (a). Motor Control ESP32 and Raspberry Pi (b)

As mentioned before, the processors are broken into three parts, a Raspberry Pi and two ESP32s. (Figure 4.24a) shows one of the ESP32s. This board uses the microphone and internal touch sensors to detect stimuli and sends its data to the Raspberry Pi to be processed. The on-board touch sensor has lines that are directly connected to the GIPO ports of the ESP to allow for its internal touch sensor readings as needed. This ESP32 also has three microphones attached to it. Powered by the on-board 3.3V power output, the microphones change in resistance depending on the volume the sense. The microphones are then connected to 20 gain operational amplifiers (op-amps) that amplify the signals of the microphones. Without the op-amps, the differences between sound levels are difficult to notice.

The Raspberry Pi is the SPI master. It receives data from the ESP32s, and based on the sensor information, makes decisions about what to do next using its AI. It then transmits motor positions for one of the ESP32s to handle (Figure 4.24b). This ESP32 handles the control of the head and neck motors. In addition, five limit switches are connected to the module that are used to define the limits of the robot's movements when it goes through its calibration phase. To prevent switch debouncing, the limit switches are in an RC debounce circuit. This prevents a faulty signal from erroneously triggering the robot's limit switch related code. According

to their data sheets, the motor controllers on the motors are rated for 5V of input voltage. However, the ESP32 is only capable of outputting 3.3V. To compensate for this difference, logic level converters were used to convert the ESP32 output voltage to that from the Raspberry Pi. However, this design proved to cause errors, and was scrapped.

4.3 | Software Design

4.3.1 | Section Introduction

Due to the complex nature of the robot and the tight timing requirements for various software components, the robot's software has been broken up between a Raspberry Pi and two ESP32s. The Raspberry Pi is a powerful embedded computer that runs a full operating system (OS). With a full OS comes the convenience of built-in support for embedded communication as well as pre-existing libraries for visual detection. However, the onboard operating system introduces an element of uncertainty when dealing with tasks that require tight timings. For these operations, we have elected to use the ESP32, a microcontroller from Espressif; ESP32s also come with existing support for embedded communication and tight motor control. Furthermore, ESP32s also have built-in libraries for embedded sensor communication, stepper motor pulse control, and GPIO control. Given that sensor data would need to be processed on the ESP32 and that motor controls need to be processed immediately, it was decided to split the responsibilities between two ESP32s; one will receive and send motor positions and the other will read in information from the sensors and send that data back to the Raspberry Pi. The combination of these two systems allow for more sophisticated calculations for our AI Architecture on the Raspberry Pi while the ESP32s allows for more precise control over the motor operations.

4.3.2 | User Experience

⊣ Personas

To better document and design the architecture from a user experience perspective, extensive work was put into writing out the various scenarios that the robot would expect to encounter as well as the expected reactions of the robot. At the start of this process, various personas were created. These are archetypes of people that were expected to interact with the robot. These personas were based on the expected circumstances that the robot would be placed in based on the stakeholder needs. Traits like age, sex, gender, and personality formed the base of these personas; furthermore, their technical background, expectations for the robot, and positive interactions were described. These details influenced how we would expect each persona to approach the robot, what the persona would expect from the robot, and what the robot represented to them. By detailing this information down, the root's behavior and the software design could be better tailored to their needs. See this example below:

Name: Laura Wilson (Fictitious) **Age**: 18 **Gender:** Female

Personality: Laura Wilson is an outgoing young lady who has a close-knit group of friends. She is an extrovert who enjoys making new friends and social interactions.
Background: Laura Wilson grew up on the West Coast of the United States in San Francisco with her mother and father. She attended the local high school where she joined the robotics club and played soccer; she maintained high grades, graduating in the top 10% of the class.
Job: High School Student
Technical Background: FIRST Robotics / VEX Robotics**,** Consumer Electronics
Purpose: Laura Wilson is here with her father, where they are touring colleges that Laura is considering applying to. She is at an admission tour where she is determining whether she wants to attend WPI as a technical college.
Expectations: When Laura Wilson is looking at the robot, she would be attentive to concepts and principles that she has yet to learn and wishes to.
Laura Wilson is also interested in the robot building process here at WPI and wants to know if it is fun.
Fulfillment: A positive situation would end with Laura Wilson being suitably impressed with the robot and interested in learning how to build a similar robot herself. Eventually, she decides WPI is an impressive university in robotics and places it in the top tier universities that she is looking at.

Figure 4.25 Software Personas Example: "Laura Wilson"

The age, gender, background, and personality traits of this persona inform us of a more extroverted young lady who maintains her friendships while also maintaining a life filled with sports, friends, and robotics. The technical background section details the persona's technical knowledge which greatly influences their expectations for the robot. In this case, Laura is familiar with consumer electronics but also has a basic knowledge of robotics through her participation in high school robotics. The purpose section dictates the reason why the person is interacting with the robot as well as decisions that the robot might influence. For Laura, this is during an admissions tour where she is determining which universities she wishes to attend in the future. The expectation sections describe what the persona is looking for in this robot; while Laura is looking at the concepts and knowledge that this robot represents, her father would be looking at the student input and resources that this robot is the product of. Finally, the fulfillment section describes the best outcome of their interaction with the robot which fulfills their expectations.

As part of this process, seven personas were created which covered each section of expected interactions: prospective students to WPI, parents of prospective students to WPI, current WPI students, WPI alumni, WPI professors, and younger children. Furthermore, these personas covered the circumstances in which the robot is currently expected to appear in— admissions tours, WPI community events, and outreach programs. These personas and their various traits influence the user stories which go more in depth of how each persona would interact with the robot. By starting with the people who will be experiencing the robot, the process becomes user-focused with the emphasis put on how people interact with the robot instead of what the creator's expectations are.
(See Appendix E for a Full List of Personas)

ㅓ User Stories

Each of the personas was then given a user story. These stories describe how the persona would interact with the robot, what the robot does in response, and the

progression of events. These events were separated into a list for each person. These lists create a positive story for the persona and help reveal not only how the person will react to the robot but also how the robot should react to the person. See this example below:

- Laura Wilson
- Laura Wilson has done her research beforehand and knows that a mascot robot exists but not its capabilities. She then seeks out the robot during an admissions tour.
- Robot notices Laura's approach, fixates on her and moves its head to get her attention.
- Laura moves around the robot trying to get a closer look at the inner mechanics.
- Robot head and neck tracks Laura's movements.
- Laura notices that the robot is tracking her and stops her movement. She then waves at the robot to see if it will respond.
- Robot head and neck follows the hand and makes Goat Noise in greeting.
- Laura says hi back to the robot.
- Robot makes goat noises in response.
- Laura reaches out her hand and touches the robot directly on the top of the head.
- Robot nuzzles upward into Laura's hand.
- Laura moves her hand around the robot's head still petting it.
- Robot nuzzles in general direction of Laura's petting.
- Laura decides that the robot is pretty interesting since it is able to respond well to her movements and that she would like to know how to build a similar robot.
- Laura asks the operator to take a selfie with the robot.
- Operator poses the robot with Laura to take a photo.
- Other guests move forward to the robot.
- Laura then has a lot more questions about the robot building process to which she asks the admissions guide or the operator.

Figure 4.26 Software User Story Example: "Laura Wilson"

This user story describes how and where Laura communicates with the robot through her verbal and physical interactions. The expected reactions from the robot are also listed out— not what the behavior currently is but rather what the behavior should be. Furthermore, these user stories also help identify behaviors that were not previously considered as well as resources and expectations outside of the robot interaction. These user stories also helped use divide up the personas into certain categories, each who interact with the robot in their own way: standard personas, who interact physically with the robot and treat it like a real goat; investigators, who attempt to Figure out the limitations and behavior of the robot; and observers, who are too shy or reserved to interact with the robot but will observe others who do.

From these user stories, three major issues were identified that had been overlooked. Firstly, the robot will need to be able to distinguish between ambient

noise and noise directed at it. It would be annoying for the robot to interrupt a nearby conversation with goat noises constantly. Secondly, an operator will always be required to monitor the robot and answer questions about both the robot and WPI. This information includes topics like school resources, why and the process in which this robot was built, names of the mechanisms and software structures, and why particular choices were made in its design. Finally, a fair portion of the personas did not physically interact with the robot as they were too shy or reserved, especially in front of a crowd. Some personas would try and research as much as possible before interacting with the robot. Therefore, it would also be wise to have some online resources that explain the mechanism, software architectures, and contains a copy of the final MQP paper. These user stories also drove the development of the robot's use cases, which formed a more concrete explanation of the expected behaviors.

(See Appendix F for a Full List of User Stories)

⊣ Use Cases

After the user stories were done, the next obstacle to tackle were the use cases. The use cases were created using the key observations from the user stories; each story was thoroughly analyzed and matched with use case sections. This allowed us to partition the use cases into seven categories: Toggle Manual Control, Manual Control, Toggle AI Mode, Seeking Behavior, Non-physical Interaction, Physical Interaction, and Sound Reactions. Use cases are a more formalized way to record and organize a list of actions defining the interactions between a user and the system to achieve a goal.

See an example below:

1: Sound Reactions
- a. User
  - i. Robot
- b. Purpose
  - i. Robot wants to engage with a person or group of persons through auditory ways.
- c. Preconditions
  - i. Robot is in AI mode.
  - ii. People are within interaction distance.
  - iii. Sound is not being made in response to a conversation.
- d. Triggers
  - i. Sound is detected.
- e. Flow of Events
  - i. Sound is detected.
  - ii. Robot makes a sound.
- f. Post Conditions
  - i. Robot is still fixated on the same person.

Figure 4.27 Software Use Case Example: Sound Reaction

Each use case is divided into 6 sub-sections: user, purpose, preconditions, triggers, the flow of events and post conditions. This particular use case describes the sound reactions of the robot. Each of the subcategories are listed and expanded out, the user being the use case user, purpose being the aim of the case, preconditions being things that have to be true for this use case to be applicable, triggers are the events which fire off the actions, flow of events being the list of actions or events that take place after the trigger, and lastly post conditions being conditions that are true after the actions are done.

In the initial document multiple problems were identified. First, we didn't have any way to trigger manual control which meant the AI mode was the only option. Secondly, the seeking behavior was not linked with the physical and nonphysical use case properly which resulted in an error with regards to the preconditions never being fulfilled for the seeking behavior. Lastly, early versions didn't have the sound and parallelization fleshed out, yet which resulted in there being no sound reactions case. All these issues were phased out as we iterated on the system.

⊣ Use Case Priority

The priority of the behaviors was decided by consulting our stakeholders' requirements and doing a feasibility analysis. There were a variety of reasons we decided to go with this ranking. Some options would be time-consuming to implement while others would be very computationally expensive. We listed the priority and the reasoning behind each rank in our use case document. A couple of examples of our ranking can be seen below.

Table 4.1 Prioritized Use Cases

Priority	Name	Use Case	Reason
5	Conversation Detection	Robot able to detect differences between conservation and directed cues	This particular behavior is important for the robot to not interrupt conversations and is the next step after sound detection. It would allow people to have conversation near the robot without interruption which sells the goat being able to only react to sound directly toward it.
7	Hand Nuzzle Behavior	Robot goes out of outstretched hand	This behavior is low priority because of the difficulty of doing it and low likelihood of people outstretching their hands. Guiding the head of the robot to a hand with only its vision is far harder than just pointing the head toward that hand. Furthermore, it was determined that a person is far more likely to just touch the head than to reach out a hand.

Conversation detection and hand nuzzle are 5 and 7 in the priority list respectively. The explanations go into detail about the difficulty of the implementation, complications and other reasons for its position amongst the list. The use cases and priority list drove the development of our AI architecture choice as well as a clear list of our requirements and needs.

(See Appendix G for a Full list of Use Cases)

(See Appendix H for a Prioritized List of Use Cases)

4.3.3 | Embedded Software

⊣ Communication Protocol Design

For our communication protocol we considered a multitude of choices and evaluated them based on our needs and our implementation requirements. The protocols discussed and analyzed were SPI, UART/serial, and I2C. Our general requirements were that more than 2 devices would need to communicate, we needed a fast data transfer method to keep our AI architecture up to date with sensor data and lastly the communication protocol could not be blocked.

We first considered UART which at first seemed like a good fit, it was robust, all our processors could use it and its data transfer rates were more than adequate at a high baud rate. It also had the advantage of having easily accessible libraries for both our devices. While it could not directly set up more than three devices, we could write a communication protocol which would take this into account and allow a three-device setup.

I2C was also a candidate, as it could be set up between more than 2 devices. Additionally, it allowed for fast data transfer between devices and it had a master slave setup allowing for one of our processors to control the flow of information. I2C also had a clock which ensured a robust method to prevent data loss or corruption during communicating. The downsides were that we would have to do extensive wiring, the libraries available were not intuitive, and lastly it was a new concept so it would take extra time to learn it to the level that a communication protocol for our robot could be set up.

SPI had many advantages, it had multiple libraries which allowed for easier integration. It also bolstered a controllable data transfer rate which could be set to high speed for fast data transfers. In addition, SPI allowed for more than two devices to be set up, in a master slave relationship which like the I2C could prove beneficial in our communication protocol design. SPI did have a few downsides though; we would have to set up clocks for it and it would require extensive testing to make it reliable.

We tested and attempted to set up all these protocols, after further analysis of the prototypes and the features and downsides we decided that SPI would fit best within our project. This decision in our design process was predicated on a multitude of factors ranging from initial prototype successes, to the extra steps that would be needed to set up our 3 processors with this protocol.

4.3.4 | AI Architecture

There are a wide variety of AI architectures that can dictate the behavior of a system, each with its own advantages and disadvantages. Each architecture structures the system using different theories of behavior. Based on the use cases, parallel functionality was not required for this project— the only case is for created noises in response to auditory signals. Other considerations are that the AI must be able to respond to unexpected circumstances. A crowd of people will not provide a controlled and stable environment. Finally, this architecture must be maintainable and extensible for future teams.

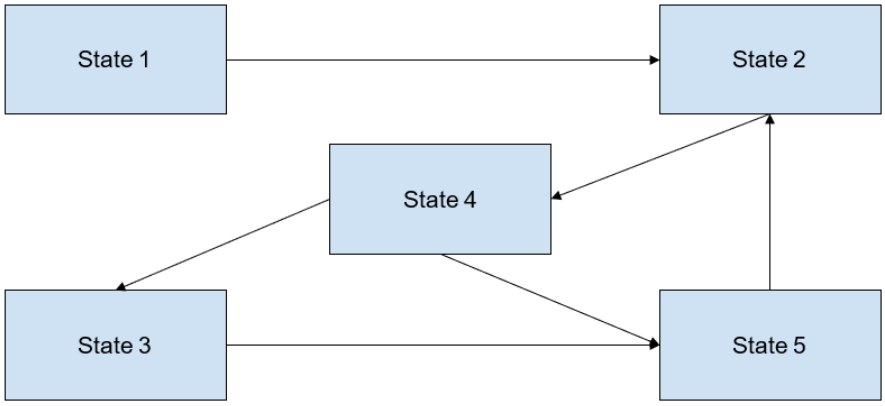


Figure 4.28 Diagram of a State Machine

A Finite State Machine (FSM) is a system that consists of transitions and states (Figure 4.3.4). Each state in this machine defines some behavior with a transition as the condition that allows the machine to switch states [41]. This architecture is simple but is useful because each state is clearly defined and allows for clear control in each state [38]. Major disadvantages include being difficult to maintain as well as producing rigid, non-goat like behavior [39]. Although further behavior could be dictated by a lower-level decision making system, that system would be one of the systems discussed in this section. Additionally, a state machine design struggles to handle unexpected inputs, preventing it from responding properly [39]; everything behavior needs to be pre-planned and every transition must be accounted for [39]. Adding to or editing this system after creation is very difficult as the many transitions must be carefully managed [41]. It is possible to use a hierarchical FSM to mitigate many of these issues on a higher level, but the issues will persist through each sub FSM.

The major reason why FSM was discounted from consideration was because of the rigidity of the system: all states and transitions need to be defined for an FSM to work well. In addition, although FSMs function well in pre-defined scenarios, FSMs are generally poor at interpreting and handling unexpected circumstances that may arise in a crowd of people. Finally, FSMs also fall short in one only trait - extensibility and maintenance. If our robot was implemented with this, we would have to fully map out its statistics and all the transitions between each state, which would take far too long to map out and make it nearly impossible to add new states

to the system. The storage requirements for this would include space for: a digital representation of the system state; the combinational logic that computes new values for state variables; and system outputs from the combined system inputs and current state variable values.

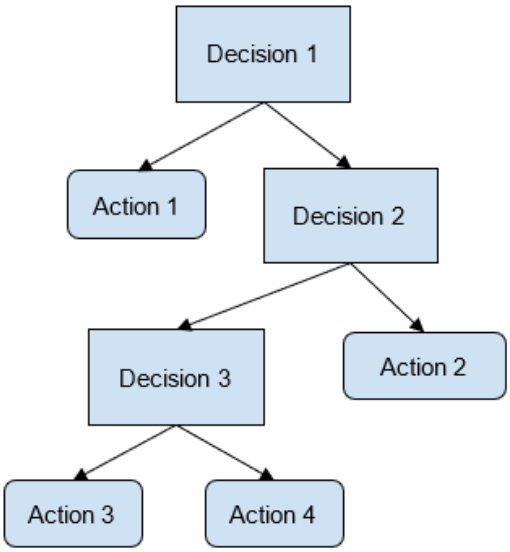


Figure 4.29 Diagram of a Decision Tree

Another simple system is a decision tree where a system parses through a tree of logical nodes. The system starts at a single root node, then moves down the tree through decision nodes. These decision nodes can then lead to other decision nodes or action nodes which can dictate behavior [41]. Like a state machine, this design is simple and allows for very defined behavior. However, it also is weak in similar areas: maintenance is difficult and decision trees are computationally inefficient [38]. Furthermore, editing or adding to this system is very complex as large decision trees are extremely difficult to understand without proper documentation [13]. As with FSMs, these problems can be mitigated using sub-decision trees but once again, the problem persists. The space requirements of decision trees are determined by the number of nodes in the tree but ultimately the space requirements are determined linearly as such would easily fit within our available storage.

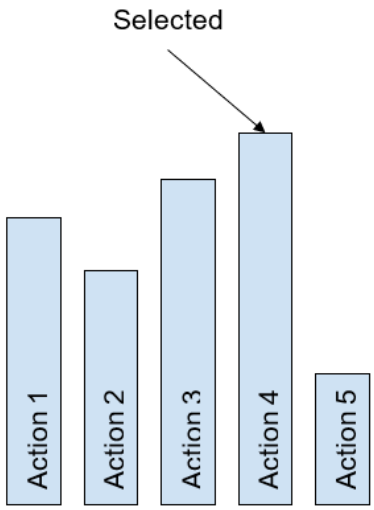


Figure 4.30 Representation of Utility Priority Score

Given those traits, decision trees were discounted from the selection because they have the same weaknesses as FSMs. If they were implemented with our project, they would contain leaf nodes as our robots' behaviors and as such everything would be based on a yes/no choice as the tree splits down. Decision trees are slightly better at handling unexpected input but not as well as other architectures listed in this section. Maintenance and extensibility are still issues; both important in a project where the code will be built and change hands many times, often without the original writer's present.

Another viable architecture is utility-based AI where external behaviors are scored based on previous actions and system input (Figure 4.3.6). Depending on those factors, the highest-scoring behavior is the one that the system deems to be the most important at each moment [41]. The advantage of this approach is that it is not as rigid as the previous two methods and still allows for well-defined behavior. This lack of rigid transitions allows utility-based AI to handle more unexpected situations [43], like those that could arise in a crowd. Its major detriment is that properly scoring the actions can be complicated and an incorrectly balanced system will not act as intended [38]. Furthermore, utility-based AI is limited in the number of exhibited behaviors as each one needs to be predefined in order to be scored [43]. If we went with this implementation, each behavior would have to be scored and balanced against all the other behaviors in such a way that there are no looped

behaviors. It would easily move along transitions and be able to act in more unexpected situations, though the scoring would need to be adjusted every time a new behavior is added. The storage would need to be enough to fit all the scores and nodes, which are linear in nature. As such, a modern processor board should have no problems storing it.

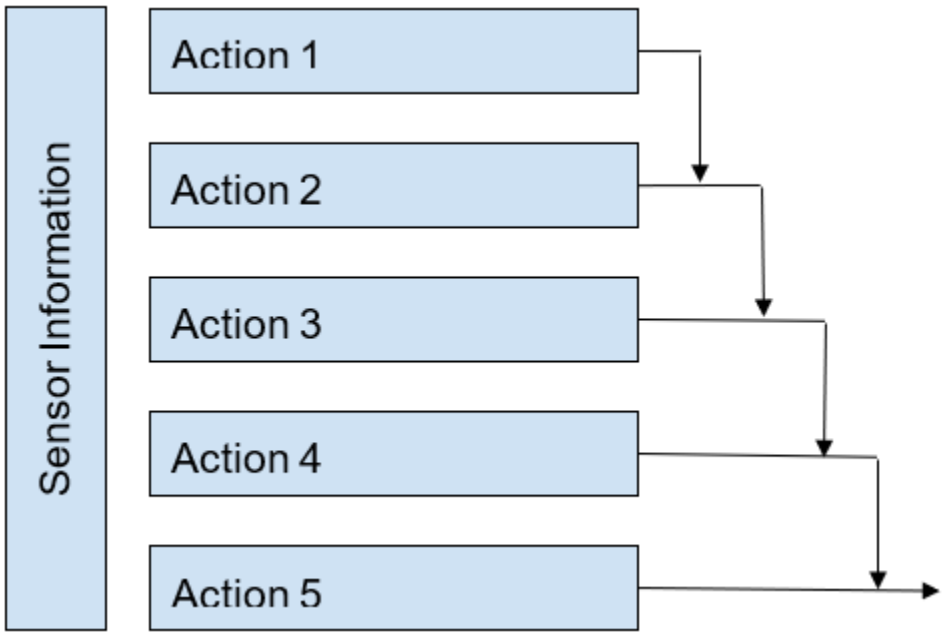


Figure 4.31 Diagram of a Subsumption System

One other behavior-dictated system is subsumption architecture. This architecture has various ordered levels (Figure 4.3.7). Each level denotes some behavior that the robot should exhibit with higher-prioritized behaviors at the bottom levels. The top-most level produces a set of external commands which are passed to the next level. Each level then modifies the commands as necessary [25]. For instance, if a robot is built to explore and map its environment, one top-level behavior would be wandering around and collecting sensor data around it. A lower level might be avoiding physical obstacles. For example, if the robot will hit a wall, this level will modify the set of commands to prevent the collision [25]. This behavior is similar to a FSM in that there is a set of predefined behavior which the robot can exhibit. However, like utility-based AI, it avoids having strictly defined transitions between behaviors [41]. Subsumption AI is also easier to maintain and

extend compared to FSMs but suffering from a different problem— scaling. As a robot needs to exhibit more and more types of behavior, it gets difficult to determine the correct order of the levels and if behaviors need to be done in sequences [25]. If we implemented this our top-level behavior would be interacting with humans, a lower level behavior would be for example nuzzling a hand. The space requirements would be minimal as subsumption architecture is built with no need for memory in mind.

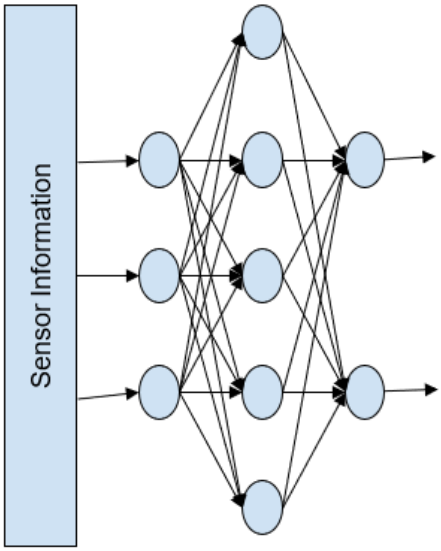


Figure 4.32 Diagram of a Neural Network

Neural networks are becoming one of the most popular methods of handling difficult to define behavior. With various layers of neurons, properly trained and designed neural networks can emulate very complex behavior. Signals are passed from neuron to neuron and then combined into an output the system can then follow (Figure 4.3.8) [41]. The major benefit of neural networks is the system being able to handle a far wider range of inputs, some of which were not accounted for by the programmer [45]. However, there are also major downsides: training a neural network is expensive and time-consuming; training requires a training set of thousands of samples to teach the neural network, which requires a large amount of work; a poorly trained network can behave in unintended ways [41]. Furthermore, the creator loses any amount of fine control over exhibited behavior—if a behavior

needs to be changed, new training material is required [45]. It is simply impossible to use this for our implementation, but if we hypothetically did use this system, we would have to make a set of goat data and train the model with it. This would allow for a unique behavior, but we wouldn't be able to add or edit things unless we wanted to fully retrain the model. The space requirements would be the weights for all the layers and the nodes themselves. As we wouldn't store any training or test data, our chip would have an easy time storing the neural network

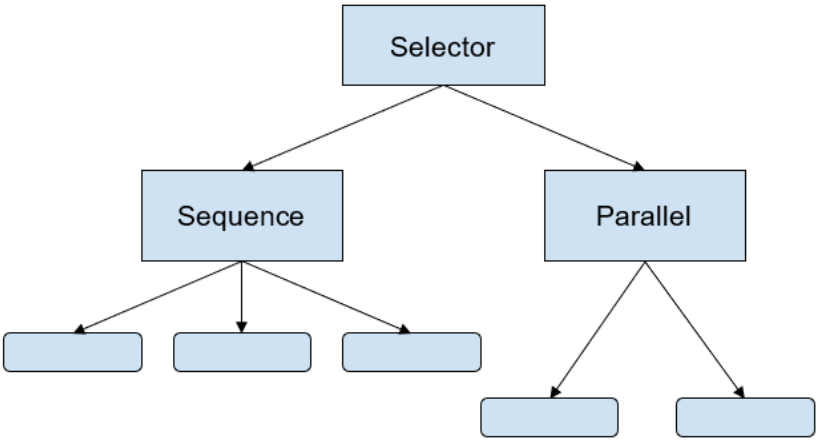


Figure 4.33 Diagram of a Behavior Tree

Behavior Trees are another way to define the output behavior of a system. In this system, behaviors are structured in a similar manner to decision trees, except the decision nodes have more abilities. In addition to decision nodes, behavior trees have sequence nodes—which dictate a sequence of behaviors—and selector nodes — which only execute one of the behaviors underneath (Figure 4.3.9) [46]. Furthermore, behavior trees also return status results from the left nodes upward; this often includes information on where the node has successfully run, cannot be run, failed in trying to run, or is still running [46]. Some of the major advantages of behavior trees are that they can dictate more behavior with less structure than decision trees and FSMs. They are also better for handling multiple known situations, such as in the example above with the opening of a door. However, they also suffer from a similar problem—handling unexpected sequences or abruptly switching exhibited behavior. For example, if we implemented this model, it would

be nearly impossible to envision and plan for every eventual situation that could happen in a crowd of people. Furthermore, the robot would be constantly switching its behavior to handle everything happening in the crowd. If a person starts petting the robot, it needs to immediately break away from tracking another person and react to the petting. In terms of storage we would need to store the entire tree— including its transitions— which may be difficult to store depending on the size. However, it should fit the tree for our implementation.

ㅓ AI Architecture Decision

The primary function of this AI Architecture will be to imitate the mannerisms and behavior of a goat such that although the robot is obviously mechanical, it reacts and functions like a well-behaved goat. From our research, visiting actual goats in a farm, we observed that goats almost always act in a single-minded manner— only focusing on one person or action at a time. Reflecting that, our intended behavior of the robot does not need major parallel functionality; the only case in which parallel behavior is required is when the robot will make goat noises in response to speech. Whatever the movement or action it is in, the robot needs to open its mouth and make a noise. Also related to its behavior, the robot must be able to exist and function in an environment where there is little to no guarantee of external factors. This robot will mainly be functioning in a crowd of people; crowds are unpredictable so the robot must be able to handle unexpected circumstances and adapt accordingly. Another factor that we considered in this decision is the maintainability and extensibility of the architecture; this robot will be passed between multiple MQP teams, as such the architecture needs to be easily modifiable so that future groups only need to add or tweak parts. This also leads to the last factor— future teams may need parallel behavior as the robot expands. Walking and interacting at the same time would require parallel behavior, so ability to support parallel behavior is important.

Many other systems, such as FSMs, were considered as well but were not viable choices. Although FSMs do not support parallel behavior, they have clearly defined transitions between each of their states. This trait would be useful in an environment where external factors can be predicted, but that is not the environment of a crowd. If a transition is not properly detected or defined, the robot will end up in a different state than it needs to be. Furthermore, as a FSM grows, the number of transitions can increase dramatically. This would make maintaining and modifying behaviors in future years difficult. Using sub FSMs mitigates some of these problems, but only shifts those issues to each sub FSM.

Decision Trees are also not viable— they lack in many of the same areas that FSMs do. Decision Trees can support parallel behavior. Like FSMs, they function poorly in unexpected circumstances and each decision must be defined beforehand. Decision Trees also suffer from the same scaling issue as FSMs; more supportive behaviors lead to more decision nodes and a larger tree. Although this can be mitigated with the use of sub Decision Trees, the problem still exists in all the trees.

Subsumption Architecture is another AI system that is no longer being considered. Subsumption Architecture does not support parallel behavior and, in fact, has traits that make modifying the architecture to support parallel behavior almost impossible without major changes [48]. Since only the most important lowest-level behavior is ever returned and no other behaviors are ever weighed against each other, it would require additional custom functionality [48]. Subsumption architecture can handle unexpected circumstances and situations, but only has a limited number of behaviors like utility-based AI. The major downside of subsumption architecture is that scaling or programming in more Subsumption Architecture is another AI system that is no longer being considered. Subsumption Architecture does not support parallel behavior and, in fact, has traits that make modifying the architecture to support parallel behavior almost impossible without major changes [48]. Since only the most important lowest-level behavior is ever returned and no other behaviors are ever weighed against each other, it would require additional custom functionality [48]. Subsumption architecture is able to handle unexpected circumstances and situations, but only has a limited number of behaviors like utility-based AI. The major downside of subsumption architecture is that scaling or programming in more complex behaviors is difficult due to the need to determine the correct order of levels and figure out sequenced behavior [48].

A Neural Network is also not a viable system for this robot to use. The prime reason why is due to the practicality of doing so, rather than the traits of the system itself. Neural Networks will already exhibit parallel behavior in its output commands, and unlike almost every other architecture discussed, it has an almost unlimited amount of different behaviors available [48]. This ability also makes a

Neural Network the best architecture for handling unexpected circumstances because it has the capability to create the appropriate behavior for the situation. Unfortunately, the practicality of creating and properly training a neural network for this project is impossible. Deep-learning Neural Networks require a large amount of training data to even get close to the correct behavior and that data just doesn't exist for goats. Furthermore, it is impossible to tune or change small parts of the exhibited behavior without retraining the neural network with new training material [29]. The requirement of training material— which would take a large amount of effort to produce—discounts this architecture simply because of the lack of time and funding required to find or create such data.

Behavior Trees are another viable system for this robot. Behavior Trees can support parallel behaviors for the robot without much modification or additional effort. Furthermore, the use of selector nodes allows the system to handle some unexpected circumstances as later nodes under that selector node allow for different behaviors. Behavior Trees do suffer from similar problems as decision trees in that scaling will result in larger trees. However, since Behavior Trees typically have more complex nodes, the resulting trees are flatter that decision trees and are therefore easier to maintain and update.

After deliberation and analysis of our requirements on a wide range of architectures, we first decided on using utility-based AI architecture. Initially, we chose this system because we found that although it does not support parallel behavior, it is one of the easiest architectures to maintain and extend. Behaviors are clearly defined in their respective areas and adding or removing behavior only requires modification of the list. This also points to utility-based architecture's second strength—there are no predefined transitions between behaviors. Although the number of exhibited behaviors is limited, the robot should exhibit the behavior closest to the current circumstance [28]. However, this ability is not without downsides, properly determining the scoring system is not a simple task; incorrectly balanced systems will not exhibit the correct behavior when required and the threshold between behavior can be difficult to determine.

After our initial choice, we consulted Professor Gillian Smith and did further research into behavior trees, deciding to switch to them. The reasoning behind this decision was because we could implement behavior trees with the functionality of utility-based systems in some of the nodes. This would result in an additional Composite Node which uses a utility score to determine the most valuable behavior. This combination allows for the parallel behavior of Behavior Trees, as well as the relative ability to handle unexpected circumstances would arise in a crowd. Furthermore, while this architecture would not be the easiest to maintain and update, if it were to be built in an intelligent manner, it would not be an insurmountable obstacle for future teams. Although this first year does not require extensive parallel behavior, future MQP groups might require that functionality and it would be wise to include that functionality if needed. With the choice of AI architecture made we moved on to the next step, the Unified Model Language (UML) diagrams.

4.3.5 | Unified Model Language (UML)

⊣ Object Diagram

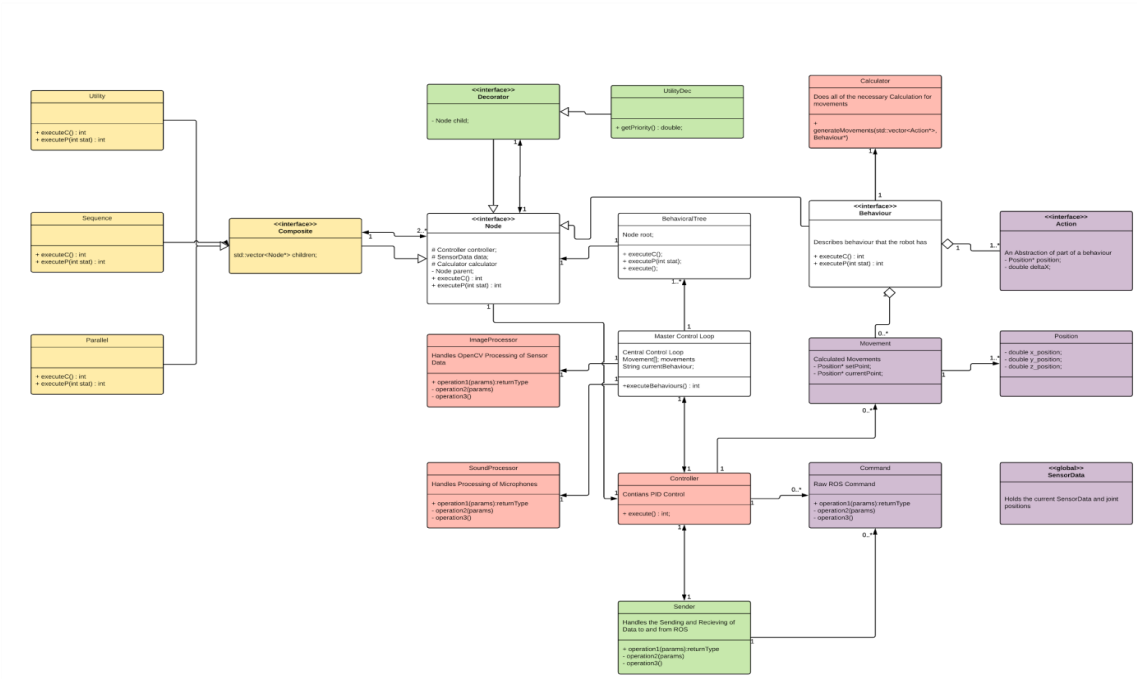


Figure 4.34 Initial Object Diagram

Figure 4.34 is the initial UML Object Diagram that was created for high level AI Architecture. This diagram shows all the expected objects and interfaces of the code as well as some of the stored variables and functions of those objects. Given that this project would have major expansions and updates in future years, the AI Architecture was designed to be deliberately abstract, with functionality distributed over multiple objects. Although these extra objects technically contain functionality that could be included in other objects, future developments may expand on those areas. The additional readability and modularity would improve the cohesion of the codebase. This object diagram also helped determine which objects would need to be shared across other objects and which objects will act just as data structures. (See Appendix I for the Full-Size Object Diagram)

⊣ Behavior Tree Diagram

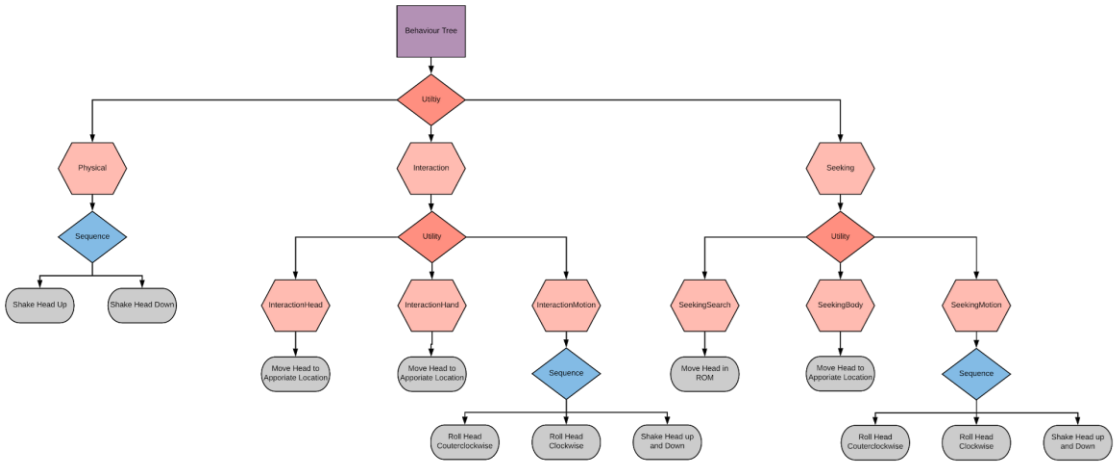


Figure 4.35 Behavior Tree Diagram

As part of the software design and a behavior tree design, a Behavior Tree Diagram was also created (Figure 4.35). This diagram details the structure of the behavior tree based on the use cases. This Behavior Tree was designed with multiple levels in mind; the first level of the behavior tree is split into each of the three main types of interaction that were described in the use cases. Utility-based composite and decorator nodes were used on this level because a utility-based system can make the decision to select which of these use cases to do under unexpected circumstances. These major types of interaction are split into another utility-based system; this second level ensures that the sub behaviors of each major interaction are not compared against each other. These subtypes of interaction then link to behaviors which will execute the intended behavior. Although the touch use case will need to interrupt any of the other behaviors, this is accounted for in the objects that manage this behavior tree. When a touch is detected, the list of nodes to execute during the next cycle is cleared and the top of the behavior tree is executed again. Given that the robot is being touched, this immediately moves the physical interaction to the highest priority.

Currently, the behavior tree relies heavily on utility composite and utility decorator nodes to determine the correct behavior to execute (Figure 4.36). There are a few sequence nodes that help define a list of behaviors that need to be

executed. Otherwise, it was determined that parallel and selector nodes were not currently required. Based on the use cases, no functionality requires selector nodes and parallel behavior is only required for audio creation. Earlier versions had parallel nodes for audio creation given that those parallel nodes were required in every branch of the behavior tree. However, it was redundant to have those nodes, since audio creation can easily be handled in a separate audio processing thread.

1. Person Head Not in Center of FOV
   a. Max 0.9
   b. Min 0.0
   c. Score based on distance of head to bottom center of vision

Figure 4.36 Decorator Design Example

Additional work was also put into the design of the decorator scores. Given that these nodes would work very similarly to utility-based architecture, it is important for these scores to be property and carefully defined. Each decorator was given an absolute minimum and maximum value as well as a general description of how the score would change in relation to the sensor information.

(See Appendix J for the Full Size Behavior Tree)
(See Appendix K for the Full Decorator Design Document)

4.3.6 | Prototype

Even with the behavior tree and a basic object diagram written out, there are a lot of implementation details which can only arise and be solved in a prototype. Therefore, the next step was to design the architecture as a whole and build a functioning prototype as a proof of concept. Firstly, there are many terms which have common meanings, but have more specific descriptions here. The first part of the AI system is like all robotic control software, namely in that there is a closed control loop that is represented in the master control loop. To determine the expected behavior at any time, the behavior tree is run. Each ending leaf in that tree is a behavior which defines a certain action or series of actions. Actions are predefined motions that the robot can do which give it life— moving in response to what it senses. For instance, the action to move the head and look at a person. These actions are used in the Calculator, which produces movements

based on known obstacles and expected endpoints. Movements are defined motions that the robot will do, i.e. move joints to these set positions in this amount of time. Movements are made up of Position, which are motor values that correspond to a certain configuration of the robot. These movements are then fed to the Controller which functions as the robot's motion controller. This Controller handles the motion planning and produces the necessary intermediate setpoints for the robot.

Another topic that was decided was the failure behavior of the composite nodes. The sequence nodes will stop upon receiving a failure, not executing any behavior yet to be executed, and returns a failure status to its parent. Parallel nodes operate similarly - if one child sends a failure status, the parallel node will return a failure status to its parent after all children have returned. Utility nodes operate in another way. When they receive a failure status from one of their children, that child is added to an ignored list. The Utility node will then recalculate scores and choose another child to execute. This process will repeat until a behavior is successfully executed or if the highest score received is zero. In both cases, the ignored list is cleared but if the highest score is zero, a failure status will be returned to its parent. This makes utility nodes able to handle some of the failures by choosing the second highest scoring behavior to execute. This functionality does add in one consideration in that if the root node is a utility node, one of its children must have a minimum score higher than zero else an infinite loop is encountered.

Another detail that was discussed was how the motion controller would handle parallel behaviors. Although the behavior tree should never produce conflicting movements, it is possible that this case happens as the behavior tree gets more complex. For example, imagine a case where the behavior tree produces two different set points under a parallel node which are in opposite directions. To handle this circumstance, there will be a movement handler which will take in the movements from the behavior tree and remove conflicting behaviors based on first-come importance. Furthermore, since the motion controller only takes in a single target position, this movement handler would combine all movements into a single movement, which only references a single target position. Finally, parallel nodes

returning a status after every return is not supported because this would result in a compounding number of statuses being passed around the behavior tree.

One further topic is the reusability of nodes within the tree— particularly with many behavior nodes sharing actions. For this functionality to work, the parents of these nodes are switched between executions so that the child will always choose the correct parent to call. However, this also adds another factor to consider in that nodes are not able to be shared underneath a parallel node because the child nodes will not know which parent to call. Take the example where there are two sequence nodes under a parallel node which both reference the same behavior node. The parallel node would call both sequence nodes which would both call the same behavior node. When the behavior node needs to call its parent node, it doesn't know which sequence node to call— especially since it would have to call its parent twice. Which sequence node called it first and does the first finished execution correspond to the first or second parent?

In addition, another topic that was sorted out were the requirements of both AI architecture, the motion and path planning, and the motor controller. This led to developments like the previous discussion on parallel behavior as well as the fact that the motion controller can only take in one end setpoint. This overall planning greatly assisted in some of the integration for the integration prototype as the roles of each system were clearly defined beforehand.

Using these considerations as well as the produced UML, a prototype version of the architecture was created. This prototype was built to only handle a single point in 3D space, operate on a CLI input scale, and doesn't communicate with third-party sources like the sensors or OpenCV yet. However, it does test the functionality of all the behavior tree nodes and is also built with abstraction so that only minor modifications are needed to apply it to a particular robot. This prototype also assisted in the integration of the AI and the messaging and motion planning as well as various requirements that are required for programming robots. By helping hash out implementation details, this prototype laid the foundation for the final version

with its clear separation of responsibilities for each object and forced the team to make decisions regarding utility nodes, parallel nodes, and conflicting movements.

5 | Prototype Testing, Results and Progress

5.1 | Mechanical Design

5.1.1 | Differential Mechanism Testing

Currently, there are two versions of the differential mechanism for the head that we are working with. Both the theory and abstract design behind them are the same. The first is a small, functional prototype. Its size meant we could quickly make low-cost fixtures and models to test functional implementation with the electrical systems and software. We used the smaller prototype for verifying joint kinematics, testing joint speeds, as well as for figuring out the cable routing path. The second prototype is a 1:1 scale model of the final design to be used for full scale testing. This assembly is also functional, though we have not yet fully implemented it. We have been able to integrate it with a partial head frame assembly, however due to the current coronavirus pandemic we do not have access to those parts, and we are unable to include images of that setup currently.

⊣ Mechanism Cabling and Routing

As discussed earlier in section 4.1.2 |, selecting the correct cable material was critical to the viability of the mechanism. The earliest prototypes used fishing line as the cabling. To work out the routing of the cables, some spare strands of TPU 3D-Printing filament were used as its high elasticity made it easier to hand tension, which became laborious while figuring out how to properly route it with a less elastic material. Obviously, this elasticity was less than ideal for the final version, however it served as a useful intermediary to practice the windings. The first larger scale prototype was initially intended to use steel wire rope; however, UHMWPE

was substituted instead, and is currently what is used to transmit power through the mechanism.

(a) Fishing Line (b) Wire Rope. (c) UHMWPE

Figure 5.1 The Three Assembled Prototype Differential Mechanisms

What we found when using a fishing line was that even though we could accommodate the minimum bend radius, the line still had too high of a bending stiffness for our purposes. Additionally, fishing line is designed to have some elasticity for its intended use case (fishing), which would reduce the overall stiffness of the mechanism at scale. Steel wire rope suffered from similar issues. Though its tensile strength was high and its elasticity very low, its stiffness caused issues with mechanism binding. In both cases, we did not use measurement tools to estimate the power losses due to the cable stiffness and friction, however it was evident from manual operation that neither material would suit our needs. UHMWPE provided a high tensile strength and low elasticity while having a relatively small minimum bend radius and low bending stiffness. Even when fully tensioned, the mechanism can still be operated manually with ease, and the major resistance comes from the friction in the pulley bearings. We did observe some minor vertical sliding of the cable as the mechanism is rotated, however due to the low coefficient of friction of UHMWPE with most materials, we assumed this sliding friction to be negligible.

⊣ Kinematics and Speed Testing

While we could conceptualize the kinematics of the differential mechanism we used, we needed to implement the mechanism in a test fixture and drive it

through software to verify. Additionally, doing so allowed us to visualize for the first time the speeds of the joints.

Figure 5.2 Servo driven test fixture

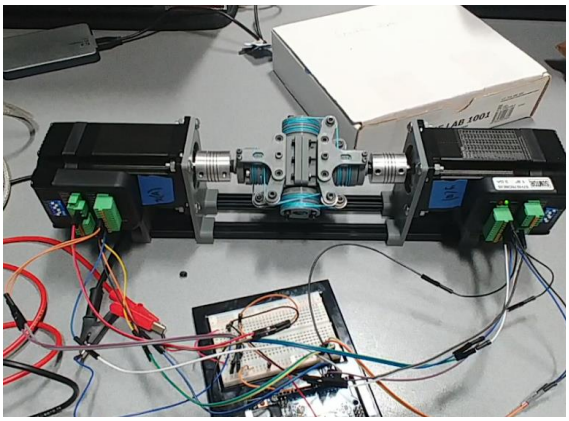


Figure 5.3 Stepper driven test fixture

The first test fixture we made (Figure 5.2) simply had two hobby servos mounted in a direct drive configuration with the mechanism. We ultimately would need more rotational range (and more power) than what these motors could offer, but it allowed us to test our kinematics. Using this, we quickly verified that our kinematic relationships were correct. A second version was later created that utilized closed loop stepper motors mounted in a similar configuration (Figure 5.3). This setup allowed us to test integration with the control software that was currently being written for the embedded microcontrollers we would be using. As a side note, while that software was in progress, a piece of open source CNC control software called GRBL was modified to accommodate our parallel kinematics. This was a quick way to ensure everything was working properly before moving on to closer software integration. Due to delays in the development process, this same modified software was again used to test the final functional prototype before switching over to our own software.

The final version used for testing was constructed to more closely model the actual robot design. This gave us a better tool to visualize joint speeds as well as what range of motion we needed. It also was intended to serve as a temporary

platform for mounting sensors and cameras so they could be tested on a functioning mechanism while further design and production work was done in parallel.

5.2 | Electrical Systems

5.2.1 | System Architecture

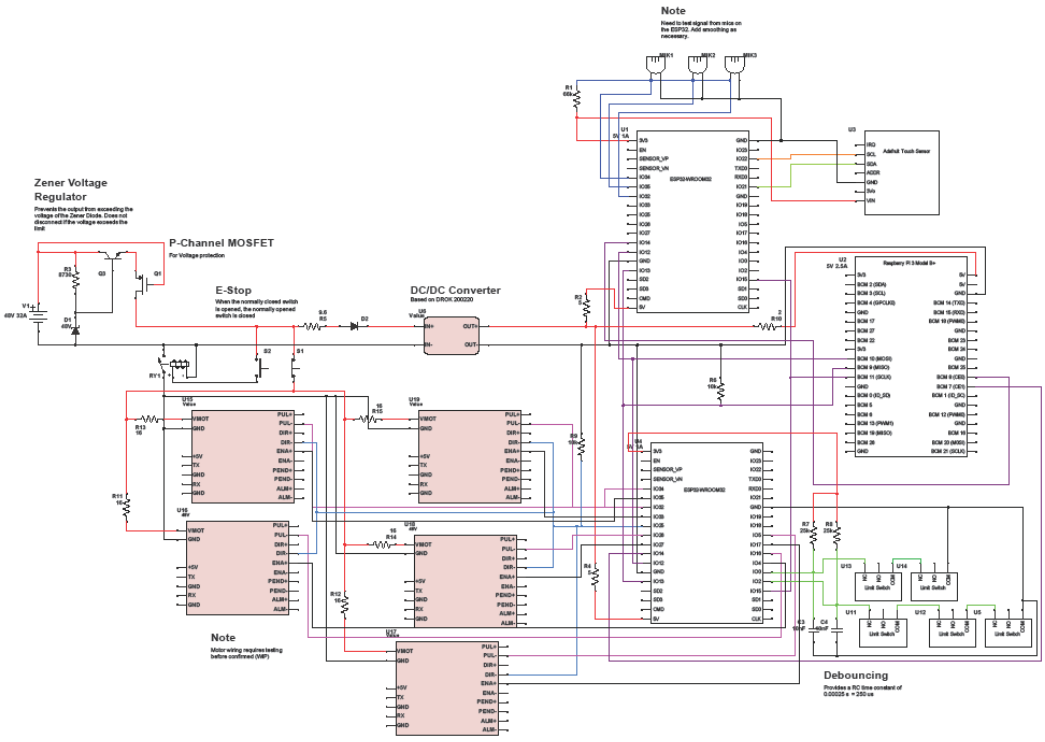


Figure 5.4 Electrical Systems Schematic

The electrical systems of the robot include power delivery, processing, sensors, as well as motors and controllers. Currently, the robot receives primary power from a residential mains voltage source which is converted to 48 volts DC, and further regulated down to the required voltages for each system component. High level processing is performed on a Raspberry Pi single board computer. Timing critical tasks such as data collection and motor control are performed on several ESP32 microcontrollers. Actuation is accomplished using stepper motors with integrated closed-loop drivers that are controlled by an ESP32.

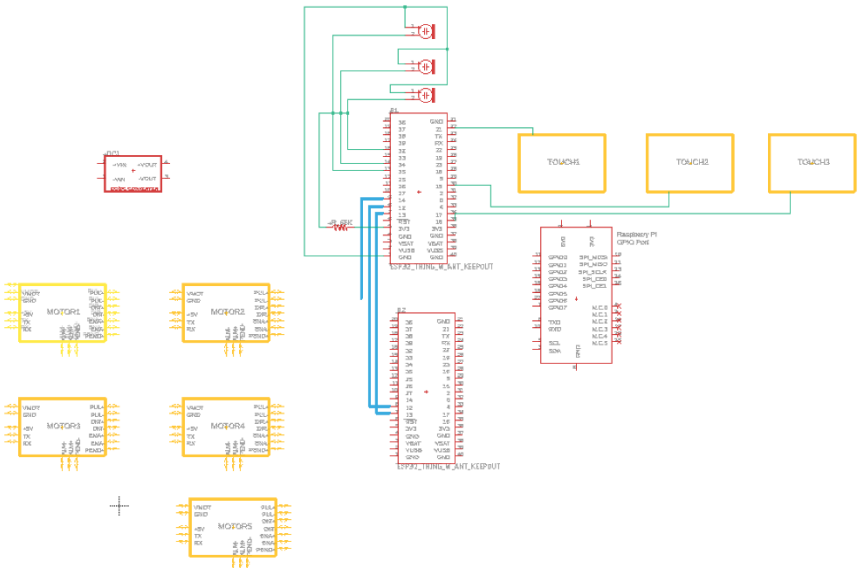
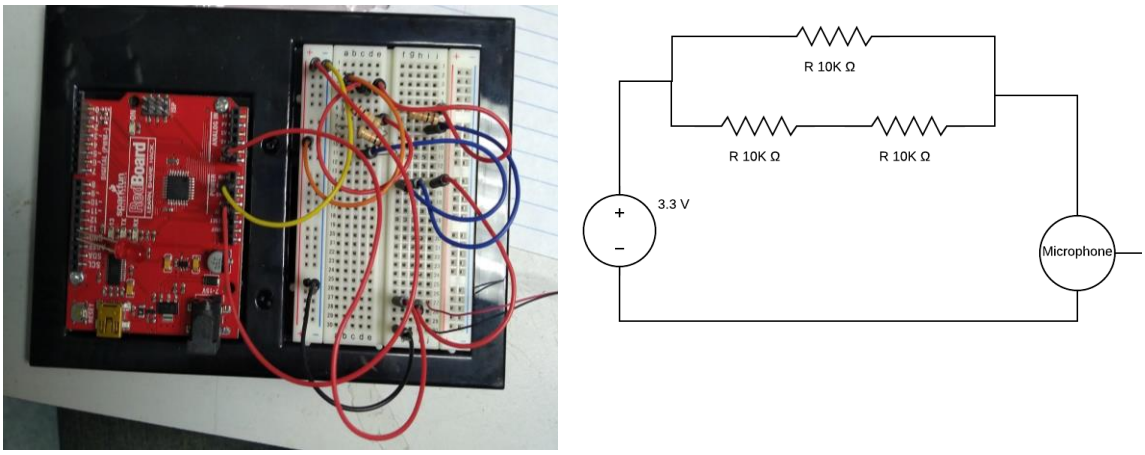


Figure 5.5 Electrical System Schematic Ported to Autodesk Eagle

As discussed in section 4.2 |, the initial electrical system diagram (Figure 5.4) was made using software provided by Digi-Key. Despite its small learning curve and integration with their parts catalogue, it was not suitable for our final design, and we ultimately moved to using Autodesk Eagle. With no prior experience with the software, we did face some setbacks. However, the utility for future teams made it a valuable effort and provided us with an opportunity to familiarize ourselves with industry standard software. Currently, the redesign (Figure 5.5) in Autodesk Eagle is not yet done but will be fully completed in the remaining project term. This delay was in part due to the coronavirus pandemic.

5.2.2 | Prototype Circuits and Testing

At this stage, we have built several circuit prototypes to test alongside our current mechanical and software prototypes. These test circuits were mainly used to check the functionality of sensors and motors. By ensuring individual functionality, later problems can be traced to implementation rather than problems with the components. Due to the coronavirus pandemic, we were unable to continue construction and testing; current progress on the prototype implementations of our electrical designs reflects those completed in C-term.



(a) **(b)**

Figure 5.6 Microphone Test Circuit with Arduino. Physical circuit(a) and Circuit Diagram (b)

The first test circuit made was for the microphone, which is read by an Arduino Uno (Figure 5.6). An ESP32 will ultimately be responsible for logging microphone data, however this was decided after this test setup was made. Different software was eventually written for the ESP32; however, the function is simple enough that any microcontroller would suffice. Like the ESP32, the Arduino Uno is equipped with 3.3V output pins that have a maximum current of 50mA. However, the microphone has an excitation voltage of 10V, so we created a voltage divider to scale this down. While this is not ideal, it sufficed for testing purposes as the microphone only pulls 0.5mA, which falls within acceptable limits after the voltage divider.

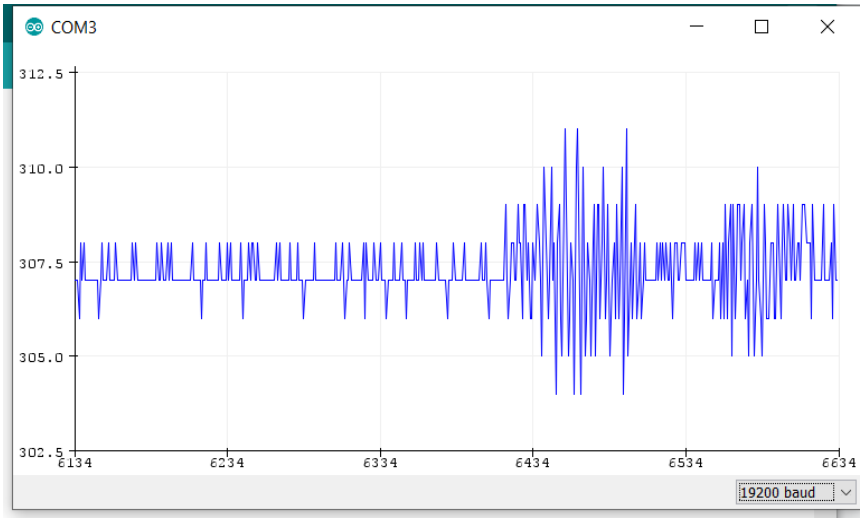
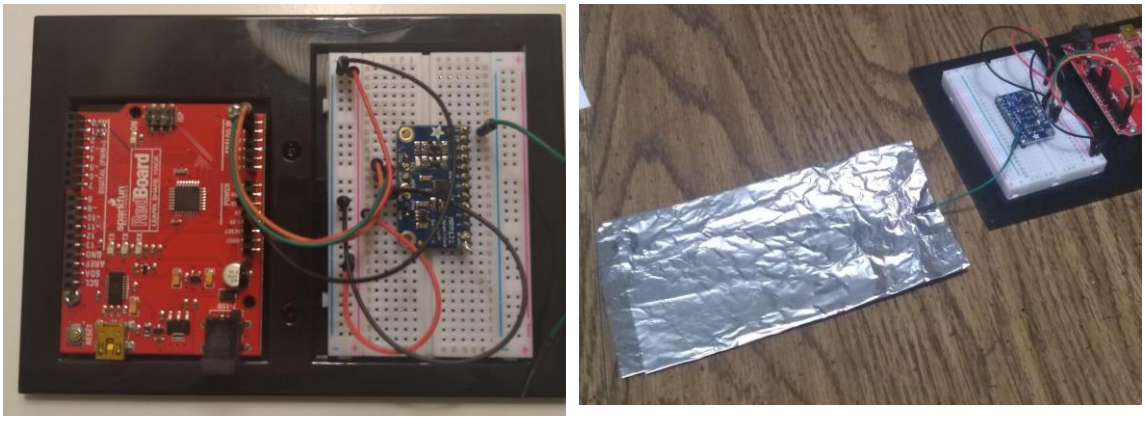


Figure 5.7 Microphone Testing Results, mV vs Seconds

To test the microphone, we performed a series of loud claps a few meters away from the microphone, at about the maximum distance we expected interaction with the robot to occur. We visualized the results of this test using the Serial Monitor and Plotter of the Arduino IDE (Figure 5.7). When there is no noise, the serial monitor oscillates around a digital value of 307.5 mV, indicating the static noise of the room. The impulse response from the claps can be seen clearly, satisfying our functionality test.



(a) **(b)**

Figure 5.8 Touch Sensor Circuit and Test Setup

The second test circuit was for the touch sensor. Like the microphone, the touch sensor was tested by connecting it to an Arduino Uno. The VIN and GND keys of the sensor were connected to the 5V source and common on the Arduino respectively. The SCL pin was connected to the I2C clock SCL pin on the Arduino, in this case A5, and the SDA pin was connected to the I2C data SDA pin on the Arduino, in this case A4 (Figure 5.8a). Finally, a wire was connected to one of the keys of the touch sensor. Using the example code that came with the Arduino library for the MPR121 controller, the Arduino would print messages to the serial monitor when a key was touched, and indicate which key was being touched. The range of the key can be extended by connecting the wire to conductive material, such as aluminum foil (Figure 5.8b). To test if the touch sensor could operate under a layer of material, an additional material was put on top of the aluminum foil. In this case, both paper and wool cloth were used. The touch sensor could detect someone touching both

materials. However, the limits of the touch sensor are still unknown. It likely will not be able to detect touch through thick material or material that is nonconductive. More intensive tests need to be done on the ESP32 to ensure the touch sensor is working properly.

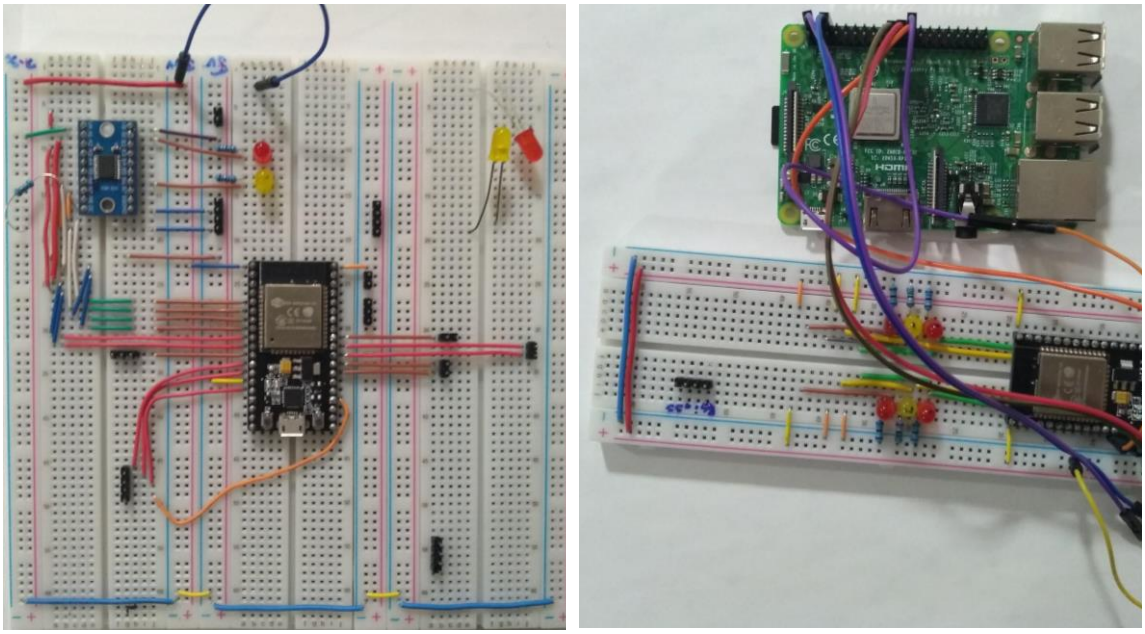


Figure 5.9 Motor Test Circuits

The final test circuit is the motor test circuit (Figure 5.9). This circuit was used to test the ESP32s ability to drive the motors after receiving commands from the Raspberry Pi. Additionally, this circuit had a dual purpose of mapping what the physical connections of the circuit would look like. The ESP32 was connected to the Raspberry Pi using their designated pins for SPI communication. From the monitor of the Raspberry Pi, position commands were sent to the ESP32. The ESP32 would then move the motor to the position. In the top left of the first circuit (Figure 5.9, left) is a logic level converter. During the testing of this circuit, we discovered that the logic level circuit was causing the motion of the motors to be erratic, causing it to fail to stop at the correct position. Once the logic level converter was removed, the motors were able to actuate properly. Using the information gained from the test circuit, the motor circuit was refined to only take a single breadboard (Figure

5.9, right). With more testing, the motors were able to move properly on the new circuit.

5.3 | Software

5.3.1 | Inter-device Communication

Serial Peripheral Interface (SPI) communication is a synchronous serial communication interface used in embedding communication. SPI communication works by using at least four pins which are wired from the master device to each slave device. The synchronous clock pin (SCLK) syncs the clocks of both devices so that each only samples the buffers at the correct time. The master out slave in (MOSI) and the master in slave out (MISO) pins are used to transfer information between the devices. As per their names, the master uses the MOSI pin to send information to the slaves and receives information from the slaves through the MISO pin. Finally, the slave select (SS) pins are used by the master to determine which slave is actively exchanging information with the master. Given that the Raspberry Pi would be communicating with multiple ESP32s, the Raspberry Pi was chosen to be the master and the ESP32s the slaves.

SPI communication sends sequences of bits from one device to another—in practical terms, these sequences can be interpreted as hexadecimal or character arrays. Therefore, a predefined communication protocol must be determined beforehand so that both devices can correctly interpret the commands. To ensure that each device is only reading in the buffers that contain commands, the buffer needs to contain a pattern before and after each command. This pattern indicates the presence of a command to the receiver and ensures that the buffer was correctly transmitted. Given that it is extremely unlikely for this pattern to appear randomly, a sequence of 10 bytes of all 1s was chosen.

Table 5.1 Control Byte Definition

Control Byte								Instruct Byte	Data Bytes
0	1	2	3	4	5	6	7	See Table 5.2	Binary Data
Dummy bit	Request Resp	Mirror Enc							

Table 5.2 Communication Command Definitions

Command	Description	Data	Data Explanation	Communication Direction
ESTOP		Boolean	Whether to Estop	BOTH
TOUCHINFO	Touch Sensor Information	Boolean Array	Whether each touch sensor is being touched	ESP32 -> RasPi
AUDIOINFO	Audio Levels for Microphones	Float Array	The noise level for each audio sensor	ESP32 -> RasPi
MOTORPOSITION	Polar Coordinates for the Robot to go to	Int8_t, Int_32t, Int8_t	Motor Index, Motor Step Position, Desired Delay	RasPi -> ESP
HANDSHAKE	Initial Protocol that Setup is complete on both ends	CHAR	Indication of Ready	BOTH
REQUESTCMD	Robot requests another command	None	None	ESP32 -> RasPi

Between the two patterns, an actual command would be contained. The first byte of the command is reserved for a control byte which contains certain flags for desired response, or whether additional mirror data is attached. Additional flags can be added as necessary to avoid corruption or desire that a command be resent. The second byte of the command contains a certain command index which determines what the command is and what data is contained in the rest of the command. For instance, if the second byte indicates that this command is a MOTORPOSITION

command, the next two bytes would be interpreted as an 8-bit integer, the next four as a 32-bit integer, and the last two as an 8-bit integer. This information would be then used for the correct execution of the command. The details of the control byte as well as the various command bytes is detailed in the following tables

After testing and integration a Communication Specifications Document was created which held all the details needed to set up our communication protocol and change it to future team's needs, this document contained the common bugs we encountered during testing, how to resolve them, links to all the relevant datasheets, links to other documentation like the ESP32 SPI protocol documentation and data like the required baud rate of the boards. Explanations of common problems our team encountered were also provided to help streamline the process for future teams allowing for a quicker and easier setup.

5.3.2 | Motor Control

For this version of the robot, we decided to use stepper motors for joint actuation in favor of more expensive closed-loop servomotors. Stepper motors function by receiving an electrical pulse that indicates a step should be taken in a certain direction. Other pins control whether the stepper motor is currently enabled, which direction the motor should turn, and whether the motor has encountered a problem. By controlling how often these pulses are sent as well as direction pin, an embedded system can control the speed and direction that the stepper motor will turn.

Given that an ESP32 is being used for motor control, it needs to be able to generate those pulses. Although it is relatively simple to create a sequence of pulses of the same period between peaks, this results in a very jarring movement. So, the pulses need to be able to accelerate and then decelerate the motor; which will result in a much smoother movement. Although it is possible to generate the acceleration and deceleration pulses whenever the ESP32 receives a MOTORPOSITION command based on the desired speed, this is computationally expensive and results in a noticeable delay before each movement is executed. To mitigate this issue, a static

set of acceleration and deceleration pulses are generated on the startup of the ESP32. Then, based on the desired delay between pulses, a certain segment of those sets is added into another sequence which is then used. This results in a process that is computationally inexpensive while keeping in the smooth movement of the robot.

These pulses are sent using one of the ESP32's built-in libraries: The Remote-Control Module Driver (RMT). This specialized library can take a sequence of pulses and is able to very quickly execute them; this operation takes a matter of microseconds and the robot is able to achieve appropriate speeds with this library. Originally, the ESP32 used a system of timers to execute a function every number of microseconds. However, this function very quickly became bloated with logic and the function was unable to execute in the required number of microseconds. In addition to allowing asynchronous execution, the RMT library allows for very precisely defined pulses periods and contains functionality for a callback function when pulse execution is done so that a REQUESTCMD command can be set back.

5.3.3 | Sensor Data Processing

⊣ Audio Processing

The AUM-5041 also was easy to set up with our ESP, by simply wiring it to ground and 3.3v which is then connected to ESP ADC port allowed a solid reading for multiple mics at the same time. Resistors and a capacitor could be used to filter and stabilize the signal, furthermore to enhance the signal and move it to the band of sounds we needed an op amp could be used to first amplify the signal, and second cut off the signal to our needed range. More testing will be needed before this is implemented. Current design simply uses a resistor and capacitor to get the signal.

⊣ Visual Processing

Unlike the touch and noise information, which is being processed on the ESP32, the visual data from the camera is being processed on the Raspberry Pi. In order to simplify object detection and avoid training a neural network, OpenCV— a library that comes with pre-trained image recognition functionality— was chosen.

OpenCV allows for very simple object detection by only requiring an image and a cascade, which defines the objects to detect. However, OpenCV does have its faults: firstly, it is computationally expensive— most of the Raspberry Pi's memory is devoted to object detection. To help reduce that cost, the images are scaled down by a factor of 2.5 so that the required processing time is reduced from 0.5 seconds to under 0.1 seconds. Another issue is that the pre-trained cascades provide inconsistent object detection. During testing, it was noticed that faces and bodies tended to pop in and out of detection despite minimal movement. To account for this, a system of visual trackers was set up. These visual trackers keep track of an object's position from frame to frame and as long as the object is seen for a proportion of the frames, then that object will be used for movement generation. This system helps filter the raw data from OpenCV and mitigates the impact of both false positives and false negatives.

5.3.4 | High Level AI

⊣ Modules and Layers

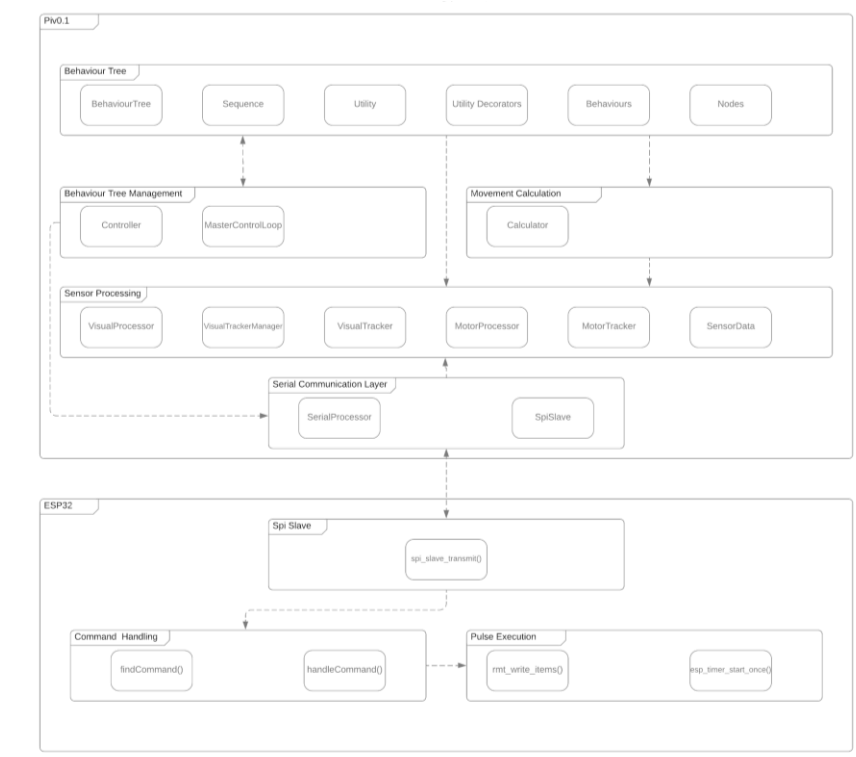


Figure 5.10 Software UML Module Diagram

The software can be divided into several layers, each with a separate and distinct task (Figure 5.10). On the Raspberry Pi side, there are currently five layers: serial processing, Behavior Tree Management, the behavior tree objects, movement calculation and generation, and sensor processing. Behavior Tree objects and Behavior Tree Management are separate from everything else because their task is to only calculate the correct behavior at a point in time and then generate a movement for it. Sensor Processing is also a distinct layer because it takes in sensor data and turns that raw data into usable information. That information is then used by the Behavior tree objects to Figure out the correct behavior as well as the Movement Calculation layer to calculate the movement. Serial Processing is responsible for encoding and decoding commands from and to the ESP32s. This is

kept separate as it only takes in generated movements and sends them to the ESP32 or puts raw data into the sensor processors.

(See Appendix L to see the Full-Size Layer Diagram)

⊣ Sequence Diagram

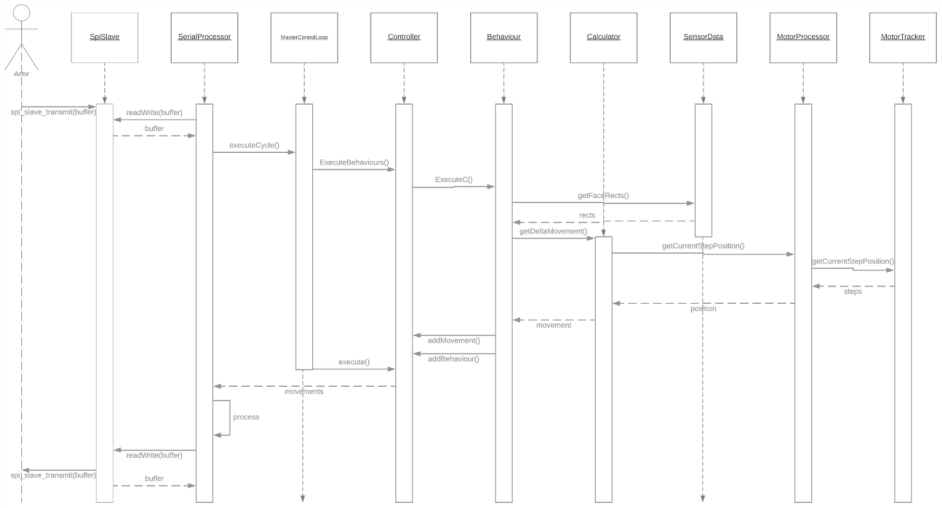


Figure 5.11 Sequence Diagram

A UML Sequence Diagram shows the progression of events and passage of data across time for a certain expected use case (Figure 5.11). In this case, the Raspberry PI Receiving a REQUESTCMD Command and generating a movement for the ESP32. The SerialProcessor reads the Command in the given buffer, moves the behaviors from the last cycle to the list to execute, and calls the MasterControlLoop to execute those behaviors. The MasterControlLoop then calls the controller to generate new Movements; the controller in turn calls behaviors in the structure of the behavior tree and the individual behavior then uses the Calculator to create a new Movement. The Calculator gets the data from the SensorData, MotorProcessor, and MotorTracker. This new Movement is then added to the Controller which then moves those Movements to the SerialProcessor which encodes the Movement and sends it to the ESP32. As stated before, this process was heavily abstracted to allow for additional functionality in the future.

(See Appendix M for the Full-Size Sequence Diagram)

This prototype version of the software architecture works mainly by creating several different threads that each handle separate tasks. Currently, there are three threads: the MasterControlLoop thread, which handles calling the behavior tree to generate movements, the SerialProcessor thread which encodes those movements to send to the ESP32s as well as read buffers back from the ESP32s, and the VisualProcessor thread which handles processing the visual images using OpenCV. Sensor and Motor data is constantly being sent from the ESP32 which is recorded and processed as necessary. This data is then used whenever a new command is requested from the ESP32 to calculate a new movement which is then sent back.

The ESP32 code works in a similar way; the ESP32 will constantly send over the current motor positions so that the latest motor position is correct. Whenever it receives a new position to go to, the ESP32 will build a list of pulse commands to be sent to the stepper motor and then send that list over. At the same time, it will calculate the expected time for the pulses to take and send a command to request a new movement when the motor has moved to the appropriate position.

This general functionality can be further expanded into expected execution of the AI Architecture as it constantly processes sensor information and creates appropriate Movements when it is needed. For the sake of explanation, assume that events happen linearly. First, the SerialProcessor reads the SPI buffer from the ESP32; it then attempts to find and decode a command. If it finds a command that is the REQUESTCMD command, it will first move the behaviors that created movements in the previous cycle to the list of behaviors to execute. Then the SerialProcessor Thread will unlock a mutex in the MasterControlLoop which will then call those behaviors in the structure of the behavior tree. This will result in other behaviors being executed; these behaviors create new Movements based on sensor data from the VisualProcessor and the MotorProcessor. These new Movements are moved to the list of Movements to send in the SerialProcessor. The SerialProcessor then sends the individual motor positions for those movements to the ESP32s over SPI.

For SensorProcessing, this sequence of events is slightly different. When the SerialProcessor Thread receives a command that contains sensor data, it decodes that data into the appropriate primitive values. In the case of the MotorData, the command is decoded into a motor index (motor identification number) and a step position for that motor. This information is then passed to the MotorProcessor which adds that information to the appropriate MotorTracker. This step position can then be accessed from the Behavior Tree and Calculator which use that data to calculate the next position to move to.

When the ESP32 receives a buffer from the Raspberry Pi, it decodes the buffer to find the command. In the instance that this is a MOTORPOSITION command, a sequence of pulses that move the desired number of steps is created and that sequence of pulses is sent off to be executed. At the same time, a timer is sent for the end of the pulse execution which will send a REQUESTCMD command back to the Raspberry Pi. Otherwise, the ESP32 will send back MOTORPOSITION commands to Raspberry Pi to indicate the current step position of all the motors.

Although these systems are not strictly necessary, additional support for writing debugging information to a log file and returning detailed status information around the behavior tree are both supported in the current codebase. There are global logger functions which allow any object to write formatted information to a time-stamped text file. Older log files are automatically deleted when there are too many old files. This allows for advanced debugging by reviewing the log file after execution, as a user can observe how the nodes of behavior tree call each other, the priority scores returned by the decorator nodes, and the raw and decoded buffers that are sent between the Raspberry Pi and the ESP32. Initially, the behavior tree would only pass an enum from node to node to indicate success or failure; following a suggestion from Professor Gennert, this was changed so status objects are passed — these are more verbose and can contain an error code, error message, and information on how to handle a failure. The combination of all these asynchronous operations and additional systems allow for a system which can continuously

request, calculate, generate, and execute movements for the robot based on sensory data that it collects.

6 | Discussion and Future Work

6.1 | Mechanical

The scope of this project is very large, and as such there is still much work to be done on the mechanical design, testing and analysis of the various subsystems. The mechanisms are currently set for testing, however work still needs to be done on a suitable neck frame. Additionally, focus was shifted away from working on the main chassis (body and torso) of the mascot, and as such will need to be completed in the future as well. Designs and a partial construction are present for the head of the robot; however, the build will need to be completed and mounted to the head joint. At that point, actuators can be mounted to drive the head in isolation, which will allow for the first performance testing of the full-scale prototype head joint. While the software is not quite ready for implementation (described below), this platform will allow for preliminary testing of the computer vision software working with real data input that reflects the coordinate transformations of the real system.

At the start of the project, we had included the outer shell and sleeve among the deliverables. The work of designing, manufacturing, and testing the silicone or cloth sleeve would have to fall to another year. One of the other aspects that will need to be designed and tested in future years is the hardened shell for the bottom of the robot which was not designed because of the prototype nature of the robot.

6.2 | Electrical

Not enough of the robot was finished in C-term to make a conclusion about the performance of the circuitry. However, there were several concerns with the design. The first major concern is the power requirements of the circuit. The circuit is designed with the intention of being run at 48V and 30A, which is equivalent to 1536W of power. 1536W is a massive amount of power for a robot, and the prices of parts that are rated for that load can be expensive and would take up a large portion of our budget. Additionally, the amount of current being used can be incredibly

dangerous if not handled properly. A major contributing factor to this high-power requirement is due to the requirements of the motors. The motors are being run at a high voltage and require approximately 6A each for safety. However, if a future team can find a way to reduce the voltage and current draw of the motors, the power requirements can be reduced. If it is impossible to do so, they should design the circuit with as many safety precautions as possible.

Additionally, the physical circuit needs to be refined further. This iteration of the project was done almost exclusively on breadboards that allowed for rapid prototyping and testing. Components and connections could be swapped out as necessary to test the circuit. However, breadboards are a poor design choice as a long-term solution. The final product should operate on a custom PCB board that connects all the components. This would require designing a permanent physical circuit board which may require parts of the circuit to be redesigned to account for the difference in hardware.

Another concern is the sensors of the robot. A major deciding factor in which sensors to purchase came from their ease of use and price. Low cost sensors with specifications that roughly matched those of the robot are chosen over more expensive parts that would better suit the needs of the robot. Due to the robot being a prototype, many parts are expected to be replaced by future teams. Because of the nature of the robot, it would have been a waste to invest in temporary parts. However, these temporary parts were not a waste, as they proved or disproved the functionality of the designs. Future teams should look at our testing and evaluate the sensors that need to be replaced and find better suited sensors for the robot.

6.3 | Software

Due to time constraints, testing of the codebase on a physical system was limited. Despite this, many of the different subsystems were manually tested. All process ending errors like segmentation faults or stack smashing had been removed so no unexpected exiting of the program should occur. SPI communication was also vigorously tested with different baud rates, corrupted buffers, and lost buffers. In all

circumstances, no exceptions were thrown. Although the buffer would be decoded incorrectly, both master and slave were able to correct their position relatively quickly. The kinematic calculations on the high-level AI were also tested and debugged. Initially, the equations were being used incorrectly and were missing some important modifiers. However, through ruthless, line by line, rubber duck debugging, these issues were resolved, and the team confirmed that the Calculator was returning the correct step positions. As the embedding system and communication was prioritized, unit tests were only recently started for the high-level AI. Although a testing architecture has been integrated with the code and makefile, unit tests have barely started, and more functions and objects need to be covered.

Further testing of the sound and touch codebase is still needed. As this code was developed at the end of term, little to no testing was conducted. While the touch sensing code was tested rigorously in lab settings, no real-world integrated testing was able to happen. Similarly, the sound codebase needs precise testing to see the sound volumes needed before amplifier circuits can be further designed for it.

The social implication of a mascot robot can be divided up into two different categories: the mascot's character and how the mascot interacts with various people. A mascot is a character, and like any character in the media, they have an influence on how people view something. Words can have multiple meanings and some words can cause unintended offense to certain groups of people. Prescribing any human characteristics to the robot automatically makes the robot represent or misrepresent those people and needs to be considered.

The actions that the mascot takes can also represent the social attitude of those people. Unintended or not, differing the behavior with different types of people can result in unintended behavior. Visual-processing software is not unbiased in that recognizing, miss-categorizing, and ignoring people of certain genders, races, or physical characteristics can happen and needs to be avoided. Any use of neural networks can be biased depending on the given training data and should be carefully considered before use. The robot must act the same to any and

all persons who interact with it and be able to accommodate individuals no matter what.

On the software side, the latest source of possible misrepresentation is the use of OpenCV for face and body recognition. The pre-trained cascades, used to identify visual features, and the training data used may contain biases that this robot will not be able to control. Another point of biases is also the priority used for calculating movements. For instance, if the robot observes two faces, which one does it prioritize? Prioritizing those faces in the center its vision leaves out people whose faces might reside in the lower or upper sections. Those excluded people could contain children who are one of the personas that were identified during the system design phase. To help counter this situation, the biases of the robot is currently set to prioritize bodies and faces from the bottom of its vision upward. Given that most of the main robot's interactions involve children, it makes sense to prioritize them.

Of course, there remains much work to be done on the software. Although most of the framework and foundation of the codebase has been written, some functionality is still missing. One of the most outstanding issues is communication support for parallel behaviors. Despite the fact that the behavior tree can support multiple behaviors being executed, the communication protocol only accounts for a single request; the ESP32 is unable to tell which command it has received or how to identify which command it has completed - it can only keep track of a single command. A system that uses a UUID for different behaviors would be able to fix that issue. Another change that would improve the codebase would be to move away from static global objects that are accessed by various objects— this trait increases the coupling of the software but allows for easier creation and assignment of objects that will be used across all objects. Although this allowed for much faster prototyping, this should be changed so that object would only keep a reference to a static object. Doing do will allow more complete unit tests to be created and the code will have overall easier maintainability, Furthermore, while visual data is being correctly processed, there is currently only bare bones processing for audio and

touch data. Despite all this future work, this year was able to create a solid foundation as well as complete much of the implementation and integration of third-party libraries.
(See Appendix N for a Full List of Future Software Work)

6.4 | General

Some aspects of the software and testing will also have to shift to future teams as a few subsystems still need to be integrated before user testing for the full robot can conclude. Additionally, future teams should look at the testing and analysis we did and decide which parts of the robot need to be replaced with more robust systems as they move out of the prototyping phase.

References

[1] S. Mohanty, “Growing Importance of Mascot & their Impact on Brand Awareness – A Study of Young Adults in Bhubaneswar City” IJCEM International Journal of Computational Engineering & Management, Vol. 17, Issue 6, Nov. 2014. [Online] Available: http://www.ijcem.org/papers112014/ijcem_112014_09.pdf [Accessed Oct 1, 2019].

[2] “Toyota Robots Help People Experience Their Dreams of Attending the Olympic and Paralympic Games Tokyo 2020,” Toyota USA Newsroom, 26-Aug-2019. [Online]. Available: https://pressroom.toyota.com/toyota-robots-help-people-experience-their-dreams-of-attending-the-olympic-and-paralympic-games-tokyo-2020/. [Accessed: 01-Oct-2019].

[3] B. Derntl, C. Regenbogen, “Empathy,” in Social Cognition and Metacognition in Schizophrenia Academic Press. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124051720000041 [Accessed: 01-Oct-2019]

[4] S. Mohanty, “Growing Importance of Mascot & their Impact on Brand Awareness – A Study of Young Adults in Bhubaneswar City” IJCEM International Journal of Computational Engineering & Management, Vol. 17, Issue 6, Nov. 2014. [Online] Available: http://www.ijcem.org/papers112014/ijcem_112014_09.pdf [Accessed Oct 1, 2019].

[5] “Toyota Robots Help People Experience Their Dreams of Attending the Olympic and Paralympic Games Tokyo 2020,” Toyota USA Newsroom, 26-Aug-2019. [Online]. Available: https://pressroom.toyota.com/toyota-robots-help-people-experience-their-dreams-of-attending-the-olympic-and-paralympic-games-tokyo-2020/. [Accessed: 01-Oct-2019].

[6] T. Hamm, et. al, “ATLAS OF GOAT ANATOMY. PART 1: OSTEOLOGY” (1970) Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/712988.pdf

[7] “Anatomy and physiology of the goat” (2017) Available:

https://www.dpi.nsw.gov.au/__data/assets/pdf_file/0010/178336/Anatomy-and-physiology-of-the-goat.pdf

[8] Page et. al. “P011 Functional degrees of freedom of neck movements: linear models may overestimate variability” (2008) Available: https://www.sciencedirect.com/science/article/pii/S0966636208700808

[9] G. Bekey, P. Koenig “AI and locomotion: horse kinematics” (1994). Available: https://go.gale.com/ps/i.do?id=GALE%7CA14822145&v=2.1&u=mlin_c_worpoly&it=r&p=PPIS&sw=w

[10] F. Zhang, et. al, “Biomimetic walking mechanisms: Kinematic parameters of goats walking on different slopes” (2018) Available: https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.4913

[11] N. Rombach, “The Structural Basis of Equine Neck Pain” (2013) Available: https://d.lib.msu.edu/etd/2978/datastream/OBJ/download/The_structural_basis_of_equine_neck_pain.pdf

[12] “Animatronics Introduction,”. Available: https://ospace.otis.edu/ganimatronics/Welcome

[13] S. Vijayagopalan, Animatronics. 2019.

[14] J. Kundig, "3-Axis Robotic Mechanisms: Animatronic Necks & Torsos", Stanwinstonschool.com, 2019. [Online]. Available: https://www.stanwinstonschool.com/tutorials/3-axis-robotic-mechanisms-animatronic-necks-torsos-part-1?utm_source=YouTube&utm_medium=JKUN%20-%203-Axis%20Mechanisms%20Animatronic%20Necks%20and%20Torsos%20Part%201&utm_campaign=On-Demand%20Course. [Accessed: 14- Oct- 2019].

[15] B. Poor, "Squash Plate Cable Controller", Poorman's Guide to Animatronics, 2019.

[16] F.J. Looft, Systems Engineering for Capstone Projects. Worcester: WPI, 2018.

[17] D. Gealy et al., "Quasi-Direct Drive for Low-Cost Compliant Robotic Manipulation", arXiv preprint, 2019.

[18] "ESP32 Overview | Espressif Systems", Espressif.com. [Online]. Available: https://www.espressif.com/en/products/hardware/esp32/overview. (Accessed: 06- Apr- 2020).

[19] "FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions", FreeRTOS, 2020. [Online]. Available: https://www.freertos.org/. [Accessed: 06- Apr- 2020].

[20] Raspberry Pi 4 Model B Datasheet, 1st ed. Raspberry Pi (Trading) Ltd., 2019.

[21] Raspberry Pi Compute Module 3+ Datasheet, 1st ed. Raspberry Pi (Trading) Ltd., 2019.

[22] M. Rowan, "Raspberry Pi 4 Hot new release – Too hot to use enclosed", Martinrowan.co.uk, 2020. [Online]. Available: https://www.martinrowan.co.uk/2019/06/raspberry-pi-4-hot-new-release-too-hot-to-use-enclosed/. [Accessed: 11- Mar- 2020].

[23] "ESP-EYE Espressif Systems | Mouser", Mouser Electronics, 2019. [Online]. Available: https://www.mouser.com/ProductDetail/Espressif-Systems/ESP-EYE?qs=%2Fha2pyFadujswH4LzMQeev06BnO64zRMg6GpEjGri4s%3D. [Accessed: 14- Dec- 2019].

[24] "ESP-EYE", Espressif.com, 2019. [Online]. Available: https://www.espressif.com/en/products/hardware/esp-eye/overview. [Accessed: 11- Nov- 2019].

[25] OV2640 Color CMOS UXGA (2.0 MegaPixel) CameraChip with OmniPixel2 Technology Advanced Information Preliminary Datasheet. OmniVision Technologies, 2006.

[26] ”Charmed Labs Pixy 2 CMUcam5 Image Sensor”2019. [Online]. Available: https://www.robotshop.com/en/charmed-labs-pixy-2-cmucam5-image-sensor.html. [Accessed: 14- Dec- 2019].

[27] Amazon.com, 2019. [Online]. Available: https://www.amazon.com/Arducam- Camera-Raspberry-Interchangeable-M12x0-5/dp/B013JTY8WY/ref=sr_1_4?keywords=m12+camera&qid=1573510

464&sr=8-4. [Accessed: 14- Dec- 2019].

[28] POM-2246L-C33-LW100-R Datasheet, 1st ed. PUI Audio, 2019.

[29] "AUM-5047L-3-LW100-R PUI Audio, Inc. | Audio Products | DigiKey", Digikey.com, 2019. [Online]. Available: https://www.digikey.com/product-detail/en/pui-audio-inc/AUM-5047L-3-LW100-R/668-1492-ND/5414022. [Accessed: 14- Dec- 2019].

[30] AUM-5047L-3-LW100-R Datasheet, 1st ed. PUI Audio, 2019.

[31] "POM-2246L-C33-LW100-R PUI Audio, Inc. | Audio Products | DigiKey", Digikey.com, 2019. [Online]. Available: https://www.digikey.com/product-detail/en/pui-audio-inc/POM-2246L-C33-LW100-R/668-1494-ND/5414024. [Accessed: 14- Dec- 2019].

[32] CMC-3015-44L100 Datasheet, 1st ed. CUI Devices, 2019.

[33] P. Sensor, "Phidgets Touch Sensor", Trossenrobotics.com, 2019. [Online]. Available: https://www.trossenrobotics.com/p/phidgets-touch-sensor.aspx. [Accessed: 14- Dec- 2019].

[34] "1129 User Guide - Phidgets Support", Phidgets.com, 2019. [Online]. Available: https://www.phidgets.com/docs/1129_User_Guide. [Accessed: 12- Nov- 2019].

[35] "Adafruit MPR121 12-Key Capacitive Touch Sensor Breakout Tutorial", Adafruit Learning System, 2019. [Online]. Available: https://learn.adafruit.com/adafruit-mpr121-12-key-capacitive-touch-sensor-breakout-tutorial. [Accessed: 12- Nov- 2019].

[36] "UCONTRO iHSS57-36-20 CNC Closed-Loop NEMA 23 Integrated Stepper Motor", Amazon.com, 2020. [Online]. [Accessed: 07- Apr- 2020].

[37] P. Khatri, "Overvoltage Protection Circuit", Circuitdigest.com, 2020. [Online]. Available: https://circuitdigest.com/electronic-circuits/overvoltage-protection-circuit. [Accessed: 09- Mar- 2020].

[38] Dill, K. (2012). Introducing GAIA: A Reusable, Extensible Architecture for AI Behavior.

[39] Guarnizo Marin, J., & Mellado Arteche, M. (2016). Robot Soccer Strategy Based

on Hierarchical Finite State Machine to Centralized Architectures. IEEE Latin America Transactions, 14(8), 3586–3596. https://doi.org/10.1109/TLA.2016.7786338

[40] An Introduction to Robot Operating System (ROS). (2017, June 26). Available: https://www.allaboutcircuits.com/technical-articles/an-introduction-to-robot-operating-system-ros/

[41] Mark, D. (n.d.). AI Architectures: A Culinary Guide (GDMag Article) « IA on AI. Available: http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/

[42] Rajesh S. Brid. (2018, October 26). Decision Trees??'A simple way to visualize a decision. Retrieved from https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb

[43] Graham, D. R. (n.d.). An Introduction to Utility Theory. Retrieved from http://www.gameaipro.com/GameAIPro/GameAIPro_Chapter09_An_Introduction_to_Utility_Theory.pdf

[44] Birrell, S. (2016, June 7). Robot Mind or Robot Body: Whatever happened to the Subsumption Architecture? – Artificial Human Companions. Retrieved from http://www.artificialhumancompanions.com/robot-mind-robot-body-whatever-happened-subsumption-architecture/

[45] A Quick Introduction to Neural Networks. (2016, August 10). Retrieved from https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

[46] Simpson, C. (2014, June 17). Behavior trees for AI: How they work. Retrieved from https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/

[47] Jirak, D., & Wermter, S. (2018). Potentials and Limitations of Deep Neural Networks for Cognitive Robots. arXiv.org. Retrieved from http://search.proquest.com/docview/2072257764/

[48] Beltran, J., & Gomez, J. (2012). Subsumption architecture for motion learning in robots. 2012 7th Colombian Computing Congress (CCC), 1–6. https://doi.org/10.1109/ColombianCC.2012.6398038

[49] R. Brewer et al., “A Friction Differential and Cable Transmission Design for a 3-DOF Haptic Device with Spherical Kinematics”, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2011.

[50] “Ultra High Molecular Weight Polyethylene Ropes (UHMWPE)”, Katradis, 2019. Available: https://www.katradis.com/high-performance-ropes/uhmwpe-ropes/uhmwpe-rope-info

[51] “Materials used for Ropes”, christiandmerchant.com, n.d., Available: https://www.christinedemerchant.com/rope_material_hmpe.html

[52] “Dyneema® in marine and industrial applications.” Pelican Rope, n.d. Available: http://www.pelicanrope.com/pdfs/DyneemaSK75_Tech_Sheet.pdf

[53] “UHMW Plastic Data Sheet,” Curbell Plastics, 2016. Available: https://www.curbellplastics.com/Research-Solutions/Technical-Resources/Technical-Resource/UHMW-Data-Sheet

[54] Muzumdar et. al., “Synthetic Fiber Capstan Drives for Highly Efficient, Torque Controlled, Robotic Applications ”, IEEE Robotics and Automation Letters, 2017. Available: https://www.osti.gov/servlets/purl/1340266

[55] “Anthrobot.com : Ross-Hime Designs, Inc : Omni-Wrist III", Anthrobot.com, 2019.

[56] Letier, Pierre & Schiele, André & Avraam, More & Horodinca, Mihaita & Preumont, A. (2019). Bowden cable actuator for torque feedback in haptic applications. Available: https://www.researchgate.net/publication/229011036_Bowden_cable_actuator_for_torque_feedback_in_haptic_applications

[57] Bajaj, Neil & Spiers, Adam & Dollar, Aaron. (2015). State of the Art in Prosthetic Wrists: Commercial and Research Devices. 2015. 10.1109/ICORR.2015.7281221. Available: https://www.researchgate.net/publication/277332478_State_of_the_Art_in_Prosthetic_Wrists_Commercial_and_Research_Devices

[58] https://ieeexplore.ieee.org/document/1642584

Appendix A: Key Stakeholder Needs

Stakeholder	Needs	SH Considerations
Athletics	The system should emote or react during sports game People can take photos of it (put their arms around it) Celebratory pose/action on goals, touchdowns, runs, etc. The system should be relatively easy to transport	The system will be a sporting events, so it should be able to interact with and hype up the crowd. The system will also have to not track debris onto the basketball courts.
Admissions	The system should interact with visitors and inform them about WPI The robot could be placed in the upcoming Bartlet Center lobby museum Visitors can take selfies with it Robot could have a fake smartphone in one of its hooves that takes selfies	The system will be on display for visitors, so it should be visually appealing and safe for visitors to approach. The system may also be used to inform visitors about WPI.
Marketing	The design of the robot should not look like any of the existing licensed Gompei designs	The robot's character should not look like the existing brand images
SAS	The system should be able to create screaming goat sounds The system should be able to act as a sound system The system should be able to launch merchandise The system should be able emote The system should have a soft exterior The system should not speak	SAS will handle the arrangements for the system. They will also have some approval over the appearance and manner of the system.

Appendix B: List of Needs and Priorities

Need	Statement	Validation	Priority
Safety	The robot should be safe to approach and operate	Robotics Department, SAS, Admissions	1
Interaction	The robot should be able to react to audio, physical, and visual stimuli	SAS, Admissions, Marketing	1
Remote Control	The robot should be able to be controlled by external controls	Athletics, SAS	1
Overheating	The robot should not overheat due to internal heating issues	All	1
Replaceable feet	The robot should have replaceable feet to avoid tracking dirt inside buildings	Athletics	2
Photo taking	The robot should be able to take photos of itself with another person.	Admissions, Marketing	2
Movement	The robot should be able to move to another location without external forces	Athletics, SAS	2
Head Movement	The robot should move its head in multiple directions	Admissions, SAS, Athletics,	2
Autonomy	The robot should have an	Admissions, SAS	2

	autonomous mode that reacts to external stimuli		
Visually Appealing	The robot should appear as a cartoonish goat not a robot.	SAS, Admissions, Athletics	2
Waterproofing	The robot should be able to handle some rain exposure	Athletics, SAS	2

Appendix C: Robot Requirements

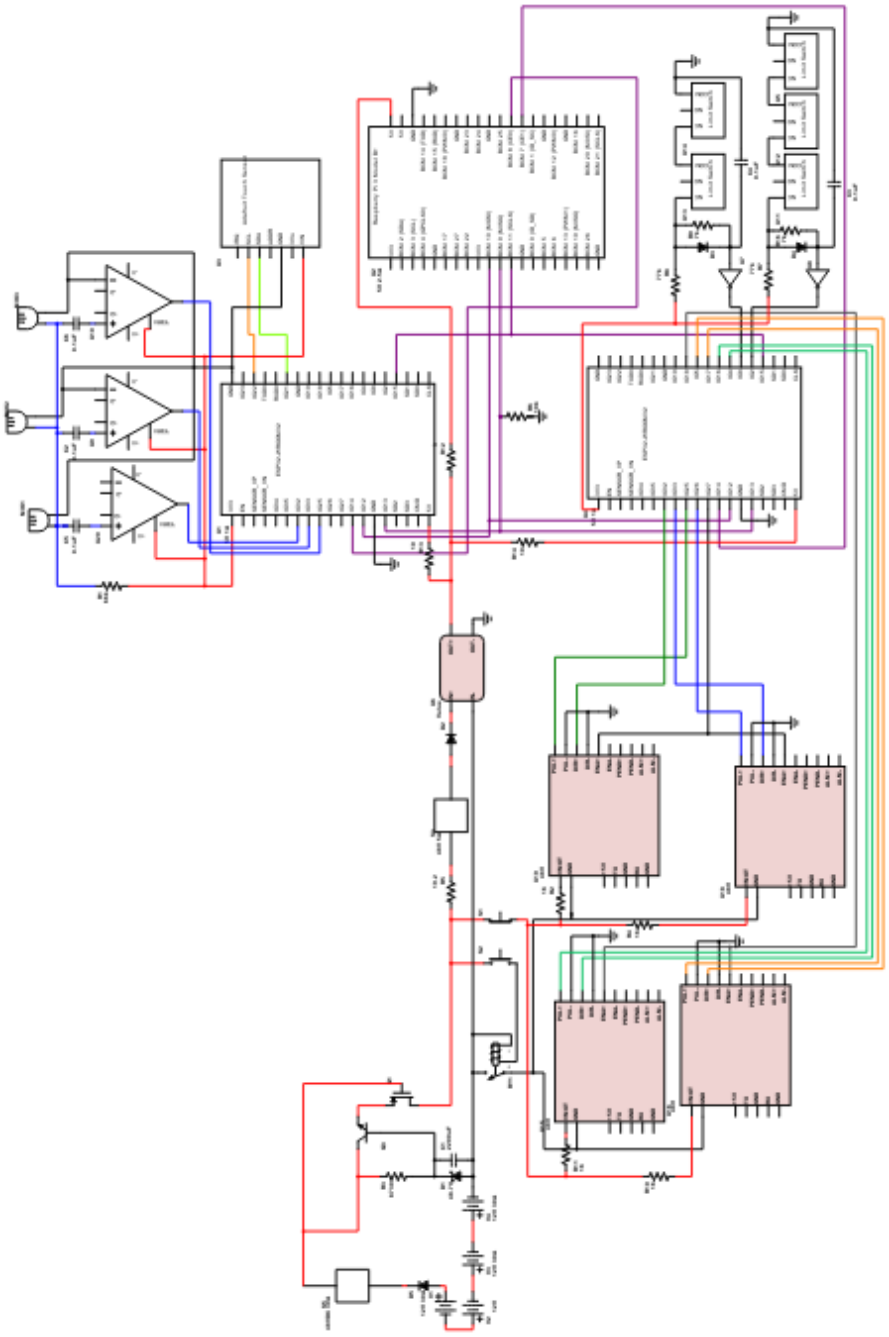
- Functional Requirements
    - The robot mascot shall be able to respond to acoustic stimuli by turning its head in the direction of the sound to an accuracy of +-10 degrees.
    - The robot mascot shall be able to respond to acoustic stimuli by producing its own sounds after acoustic stimuli made in conversational tone within 2m.
    - The robot mascot shall be able to respond to visual stimuli by turning its head in the direction of the visual movement to an accuracy of +-10 degrees.
    - The robot mascot shall be able to account for physical stimuli by being able to accommodate unpowered movement of the head to 10cm in any direction.
    - The robot mascot shall be able to pose itself into at least two different positions: Proud Goat and Charging Goat.
    - The robot mascot shall be capable of identifying people within 1 meter of its range of vision.
    - The robot mascot shall be capable of following a handheld out within 1 meter of its range of vision.
    - The robot mascot shall be capable of responding to being touched on its head with 9N of force.
    - The robot mascot shall be able to recognize when a movement is being resisted and stop movement accordingly.
    - The robot mascot shall be able to move its head and neck with a combined 5 degrees of freedom.
    - The robot mascot shall have a neck pitch range of ±30 degrees.
    - The robot mascot shall have a neck yaw range of ±50 degrees.
    - The robot mascot shall have a head roll range of ±30 degrees.
    - The robot mascot shall have a head pitch range of ±10 degrees.

    - The robot mascot shall have a head yaw range of ±40 degrees.
    - The robot mascot shall move using a quadrupedal leg system.
    - The robot mascot shall be able to produce its own sounds after acoustic stimuli made in conversational tone within 2m.
- Non-Functional Requirements
    - The robot mascot shall be able to operate for no less than 2 hours before charging or battery replacement.
    - The robot mascot shall be capable of manual control via a remote control, as well as autonomous interactions with people.
    - The robot mascot shall be water resistant to handle light exposure to rain without sustaining internal damage, long enough to be moved from the far end of the WPI athletics field to the athletic center.
    - The robot mascot shall have adequate thermal regulation to prevent overheating.
    - The robot shall be safe to approach and operate.
    - The robot shall have replaceable feet for indoor and outdoor activities.
    - The robot mascot shall use a combination of a camera, microphones, and capacitive touch sensors to sense and interact with the world around it.
    - The robot mascot shall have enough power to operate all motors and sensors at full power.
    - The robot mascot shall utilize a rechargeable battery for non-tethered operation.
    - The robot mascot shall have a port to plug in a cable to charge its battery.
    - The robot mascot shall be capable of operating while tethered and not tethered.
    - The robot mascot shall be able to operate for no less than 2 hours before charging or battery replacement. *
    -

- Derived Requirements
    - The robot mascot shall be able to fit into a standard freight or medical elevator (2300cm x 1100cm source: http://www.mitsubishielectric.com/elevator/products/basic/elevators/nexiez_mr/pdf/hospital.pdf).
- Standards Requirement
    - The robot mascot shall not be able to pinch anyone who is touching the neck or head areas during movement.
    - The robot mascot shall be able to recognize when a movement is being resisted and stop movement accordingly.
    - The robot mascot shall have an easily reachable and easily pushable button that immediately cuts power to the robot mascot.
- Software Requirements
    - The robot mascot shall be able to be controlled remotely through a web server or through a Bluetooth controller.
    - The robot mascot shall be programming in an easily readable standard with at least one comment in Doxygen standard for every file, method, and function.
    - The robot mascot shall have an interface such that it is relatively simple to add support for a new payload.
    - The robot mascot shall be able to respond within 1 sec of any given input. (Should be far less).
    - The robot mascot shall be able to be controlled through a wireless controller.
    - The robot mascot shall be able to interrupt precious commands given certain sensor input.
    - The robot mascot shall be able to have a mode that allows it to automatically interact to stimuli from external sensors.
    - The robot mascot shall have a kill switch on the wireless controller or any interface where it can be controlled.

    - The robot mascot shall have programming with unit tests that cover the most functions and situations.
- Discovered Requirements
    - The robot shall interact and respond to acoustic, physical, and visual stimuli

Appendix D: Circuit Diagram



Appendix E: Personas

1. Laura Wilson
    1.1. Age: 18
    1.2. Gender: Female
    1.3. Personality
        1.3.1. Laura Wilson is an outgoing young lady who has a close-knit group of friends. She is an extrovert who enjoys making new friends and social interactions.
    1.4. Background
        1.4.1. Laura Wilson grew up on the West Coast of the United States in San Francisco with her mother and father. She attended the local high school where she joined the robotics club and played soccer; she maintained high grades, graduating in the top 10% of the class.
    1.5. Job
        1.5.1. High School Student
    1.6. Technical Background
        1.6.1. FIRST Robotics / VEX Robotics
        1.6.2. Consumer Electronics
    1.7. Purpose
        1.7.1. Laura Wilson is here with her father, where they are touring colleges that Laura is considering applying to. She is at an admission tours where she is determining whether she wants to attend WPI as a technical college.
    1.8. Expectations
        1.8.1. When Laura Wilson is looking at the robot, she would be attentive to concepts and principles that she has yet to learn and wishes to.
        1.8.2. Laura Wilson is also interested in robot building process here at WPI and wants to know if it is fun.

1.9. Fulfillment

1.9.1. A positive situation would end with Laura Wilson being suitably impressed with the robot and interested in learning how to build a similar robot herself. Eventually, she decides WPI is an impressive university in robotics and places it in the top tier universities that she is looking at.

2. David Wilson

2.1. Age: 48

2.2. Gender: Male

2.3. Personality

2.3.1. David Wilson is more reserved and cautious than his daughter. He is also more of an analytical man who trusts statistics and math more than emotions.

2.4. Background

2.4.1. David Wilson grew up in the US Midwest in a rural location. He graduation a middle of the road university before moving to San Francisco and settling down to raise a family.

2.5. Job

2.5.1. Financial Analyst Contractor

2.6. Technical Background

2.6.1. Consumer Electronics (Smartphone, personal computer)

2.6.2. Microsoft Office Use

2.7. Purpose

2.7.1. David Wilson is here with his daughter Laura to determine if WPI is able to give his daughter an education that will enable her to get a decent job upon graduation.

2.8. Expectations

2.8.1. While David Wilson is looking the robot, he would be looking for concepts that he had research prior to the trip. He is looking to make sure that the robot is in line with current industry trends and that by building a robot similar, his

daughter would learn what she needs to know. David would be also interesting the resources provide to his daughter and also compare that to other universities.

2.9. Fulfillment

2.9.1. A positive situation would end with David Wilson observing current robotic concepts in action. David would then decide that WPI is able to give his daughter an education that would teach his daughter the proper skillset to get a decent job.

3. John Caben

3.1. Age: 31

3.2. Gender: Male

3.3. Personality

3.3.1. John is more of an introvert and spends most of his time gaming with a small group of online friends. He keeps casual work acquaintances but is mostly a solitary person.

3.4. Background

3.4.1. John grew up in the suburbans of Boston. He attended WPI, graduated, and moved to Seattle to work for Boeing. He does not have a significant other.

3.5. Job

3.5.1. Mechanical Engineer at Boeing

3.6. Technical Background

3.6.1. WPI Mechanical Engineer Degree.

3.6.2. Power User of personal electronics

3.7. Purpose

3.7.1. John has returned to WPI area for his 10-year reunion and during the party sees the robot.

3.8. Expectations

3.8.1. While John Caben is looking at the robot, he is comparing the robot with what he did during his WPI years as well as his current industry experience.

3.9. Fulfillment

3.9.1. A positive situation would end with John Caben being suitability impressed with the robot and that is demonstrates more knowledge than he learned during his WPI days and perhaps similar knowledge that he is currently applying.

4. Hannah Williams

4.1. Age: 35

4.2. Gender: Female

4.3. Personality

4.3.1. Hannah Williams is more of an introvert but does maintain a close circle of friends among her colleagues.

4.4. Background

4.4.1. Hannah Williams is a first-generation immigrant who moves from Europe during her younger years. She attended university in the US and obtained her PHD at Georgia Tech in Robotics.

4.5. Job

4.5.1. Associate Professor

4.6. Technical Background

4.6.1. PHD in Robotics

4.7. Purpose

4.7.1. Hannah Williams was passing by a university event in the quad and noticed the robot. She then takes a look to examine and judge the robot

4.8. Expectations

4.8.1. While Hannah Williams is looking at the robot, she is taking a more critical view of it. She wants to see that concepts that she is teaching are being applied and likely to ask more complex questions about the decision making of the mechanics.

4.9. Fulfillment

4.9.1. A positive situation would end with Hannah Williams being suitability impressed with the robot, seeing that it does contain and was built with her lessons in mind.

5. Betsy Ericson
   5.1. Age: 8
   5.2. Gender: F
   5.3. Personality
      5.3.1. Betsy Ericson is a rambunctious, energetic child who tends to touch whatever she wants. This has led to a tendency to accidentally break some of her more fragile toys.
   5.4. Background
      5.4.1. WEST PHILADELPHIA BORN AND RAISED. The Ericson family all live together in a single household with her grandparents; they are more economically disadvantaged.
   5.5. Job
      5.5.1. Elementary School
   5.6. Technical Background
      5.6.1. LEGO
      5.6.2. Consumer Electronics (iPad, Computers, Video games)
   5.7. Purpose
      5.7.1. Betsy is attending a summer camp near WPI and they visit WPI for a tour. At one station, she sees this cool robot.
   5.8. Expectations
      5.8.1. While Betsy Ericson is interacting with the robot, she expects it to interact with her and be cool. She is also likely to interact physically with it in unexpected ways,
   5.9. Fulfillment
      5.9.1. A positive situation would end with Betsy Ericson directly interacting with the robot in some way that she considers cool as well as not break under unexpected circumstances.
6. Sam Ayam

6.1. Age: 17

6.2. Gender: Nonbinary

6.3. Personality

6.3.1. Sam is introverted who doesn't have a close circle of friends, instead they are more of a social loner. Although they were involved in their high school marching band, they only had a group of acquaintances.

6.4. Background

6.4.1. Sam grew in Seattle, among the northern suburbans. They started music early with the French horn, which they continued throughout high school. They were raised by a single working mother.

6.5. Job

6.5.1. High School Student

6.6. Technical Background

6.6.1. Intro Computer Science Course

6.6.2. Computer Science Hobby Projects

6.7. Purpose

6.7.1. Sam is visiting WPI outside of an admissions tour, rather they are being shown around by their cousin who is attending. Sam is determining whether they want to attend WPI as a technical college.

6.8. Expectations

6.8.1. While Sam Ayam is interacting with the robot, they are more curious about the engineering process in which it is built. They would be interested in the various programming aspects as well as how students got involved in this project.

6.9. Fulfillment

6.9.1. A positive situation would end with Sam Ayam, deciding WPI is an impressive university in robotics and places it in the top tier universities that Sam is looking at. They would also respect

how the robot was built and enjoy that at WPI students are pushed to do MQPs.

7. Axle Lough
    7.1. Age: 18
    7.2. Gender: M
    7.3. Personality
        7.3.1. Axle is a relatively extroverted student who played basketball through middle and high school. He has a close group of friends and they hand out often.
    7.4. Background
        7.4.1. Axle grew up in the center of Los Angeles; both of his parents work in non-technical fields, but Axle is now considering a STEM career.
    7.5. Job
        7.5.1. High School Student
    7.6. Technical Background
        7.6.1. High School Science Courses
        7.6.2. Consumer Electronics
    7.7. Purpose
        7.7.1. Axle is here on an admissions tour to determine two points: if STEM fields are right for them and if WPI is the right school for STEM.
    7.8. Expectations
        7.8.1. While Axle Lough is interacting with the robot, he would like to see if WPI interests him at all and whether he might be able to do what he sees eventually.
    7.9. Fulfillment
        7.9.1. A positive situation would end with Axle Lough deciding that STEM is cool, want to go into a field and that WPI is the university to go to for that field.

Appendix F: User Stories

1. Laura Wilson
    a. Laura Wilson has done her research beforehand and knows that a mascot robot exists but not its capabilities. She then seeks out the robot during an admissions tour.
    b. Robot notices Laura's approach fixates at her and moves its head to get her attention.
    c. Laura moves around robot trying to get a closer look at the inner mechanics.
    d. Robot head and neck tracks Laura's movements.
    e. Laura notices that the robot is tracking her and stops her movement. She then waves at the robot to see if it will respond.
    f. Robot head and neck follows hand and makes Goat Noise in greeting.
    g. Laura says hi back to the robot.
    h. Robot makes goat noise in response.
    i. Laura reaches out hand and touches the robot directly on the top of the head.
    j. Robot nuzzles upward into Laura's hand.
    k. Laura moves her hand around the robot's head still petting it.
    l. Robot nuzzles in general direction of Laura's petting.
    m. Laura decides that the robot is pretty interesting since it is able to respond well to her movements and that she would like to know how to build a similar robot.
    n. Laura asks operator to take a selfie with the robot.
    o. Operator poses the robot with Laura to take a photo.
    p. Other guests move forward to the robot.
    q. Laura then has a lot more questions about the robot building process to which she asks the admissions guide or the operator.
2. David Wilson

a. David Wilson notices robot since his daughter mentioned it and comes to investigate.
b. Robot notices David's approach and makes head and neck movements and goat noise to try and get attention.
c. David Wilson examines robot from a relatively stationary position with small head and neck movements.
d. Robot follows small movement for 5 seconds before losing fixation and moves to another person.
e. David Wilson observes interactions of the robot with a third person.
f. David Wilson asks question to admissions guide about student resources that enable them to build a robot. David Wilson also comparing the robot silently to robot that he has seen in the news and media.

3. John Caben
    a. John Caben notices robot at the reunion and decides he wants to take a look at what students are WPI have been doing. So, he approaches it.
    b. John Caben first examines robot from a stationary position with relatively small head movements
    c. Robot head and neck tracks small movements.
    d. John Caben notices these small movements and decides to test the limits of the head and neck angles of freedom.
    e. John Caben first tests the extreme left and right movements before testing up and down.
    f. Robot tracks John Caben's head as much as it can but does not go past its limits.
    g. Operator notices John Caben's attempts and activates demonstrate sequence to show off the entire ranges of motion and all actuated movement.
    h. John Caben asks to touch the robot.
    i. Operator confirms that the robot is made to be touched and suggests reaching out hand.

j. Robot head goes to hand and nuzzles it.
k. John Caben then observes the robot and tries touching it in various spots on the head, neck, and flanks.
l. Robot makes goat noise and move head and neck in response to begin touched.
m. John Caben then steps back and questions the operator on the various systems or mechanisms that were used on the robot.
n. Operator peeling back the outer latex skin to show the inner workings.
o. Operator switches the robot to manual mode and demonstrates the physical interworking of the robot.
p. John Caben thanks operator for the closer look and steps back to tell his friends to come look at this.

4. Hannah Williams
    a. Hannah Williams has heard of the robot from her peers and seeks it out during a social event on the quad.
    b. Hannah Williams first examines robot from a stationary position with relatively small head movements
    c. Robot head and neck tracks small movements.
    d. Hannah Williams notices these small movements and decides to test the limits of the head and neck angles of freedom.
    e. Hannah Williams first tests the extreme left and right movements before testing up and down.
    f. Robot tracks Hannah Williams's head as much as it can but does not go past its limits.
    g. Operator notices Hannah Williams's attempts and activates demonstrate sequence to show off the entire ranges of motion and all actuated movement.
    h. Hannah Williams asks to touch the robot.
    i. Operator confirms that the robot is made to be touched and suggests reaching out hand.

j. Robot head goes to hand and nuzzles it.
k. Hannah Williams then observes the robot and tries touching it in various spots on the head, neck, and flanks.
l. Robot makes goat noise and move head and neck in response to begin touched.
m. Hannah Williams then steps back and questions the operator on the various systems or mechanisms that were used on the robot.
n. Operator peeling back the outer latex skin to show the inner workings.
o. Operator switches the robot to manual mode and demonstrates the physical interworking of the robot.
p. Hannah Williams also has questions about why certain design decisions were made but the operator is unable to answer them.
q. Hannah Williams thanks operator for the closer look and steps back thinking about the mechanisms that she had observed.

5. Betsy Ericson
    a. Betsy Ericson notices the robot is interacting with another group of children.
    b. Betsy runs over to them to get a closer look but accidentally stumbles into one of the legs.
    c. Robot remains balanced and interacts with another child that had its fixation.
    d. Impatient with the robot making goat noise with another child, Betsy grabs the head of the robot and brings it in front of her face.
    e. Robot compensations for unexpected movement and moves fixation to Betsy.
    f. Betsy then makes noises and strokes it for a period of time.
    g. Betsy then loses interest but still thinks that robot was cool and runs away, accidentally tugged sharply on the head.
    h. Robot remains balanced.
6. Sam Ayam

a. Sam Ayam notices the robot at a social event on the quad and comes over to investigate.
b. Sam Ayam remains apart from the WPI students but does observe their interactions with the robot.
c. Sam Ayam attempts to discern how it was programming and looks for various flaws in its behavior.
d. Being socially anxious, they do not approach the operator with their questions but ask their cousin, but he doesn't know. Instead they resolve to do their research about the robot later online.

7. Axle Lough
    a. Axle Lough meets the robot during an admissions tour.
    b. Robot notices Axle approach fixates at him and moves its head to get his attention.
    c. Axle moves around robot trying to get a closer look at the inner mechanics.
    d. Robot head and neck tracks Axle's movements.
    e. Axle notices that the robot is tracking him and stops his movement. He then waves at the robot to see if it will respond.
    f. Robot head and neck follows hand and makes Goat Noise in greeting.
    g. Axle says hi back to the robot.
    h. Robot makes goat noise in response.
    i. Axle asks operator to take a selfie with the robot.
    j. Operator poses the robot with Axle to take a photo.
    k. Other guests move forward to the robot.
    l. Axle then has a lot more questions about the robot building process to which he asks the admissions guide or the operator.

Appendix G: Use Cases

1. Toggle Manual Control
    1.1. User
        1.1.1. Operator
    1.2. Purpose
        1.2.1. Operator wants to toggle the robot to manual mode, in order to pose the robot for a photo.
    1.3. Preconditions
        1.3.1. Robot is in AI mode.
    1.4. Triggers
        1.4.1. Operator's intent to manually move the robot.
    1.5. Flow of Events
        1.5.1. Operator hits a button on the controller.
        1.5.2. All AI operations cease.
    1.6. Post Conditions
        1.6.1. Robot is in Manual Mode.
        1.6.2. No automated movements - no AI control routines are run.
2. Manual Control
    2.1. User
        2.1.1. Operator
    2.2. Purpose
        2.2.1. Operator is able to individually control all degrees of freedom using the controller, in order to pose the robot for a photo.
    2.3. Preconditions
        2.3.1. Robot is currently in manual mode.
    2.4. Triggers
        2.4.1. Operator's intent to pose the robot.
    2.5. Flow of Events
        2.5.1. Operator manipulates the left joystick to the left direction.
        2.5.2. Robot base of neck moves in the negative yaw direction (left).

2.5.3. Operator manipulates the left joystick to the right direction.
2.5.4. Robot base of neck moves in the positive yaw direction (right).
2.5.5. Operator manipulates the left joystick in the up direction.
2.5.6. Robot base of neck moves in the positive pitch direction (up).
2.5.7. Operator manipulates the left joystick in the down direction.
2.5.8. Robot base of neck moves in the negative pitch direction (down).
2.5.9. Operator manipulates the right joystick to the left direction.
2.5.10. Robot base of head moves in the negative yaw direction (left).
2.5.11. Operator manipulates the right joystick to the right direction.
2.5.12. Robot base of head moves in the positive yaw direction (right).
2.5.13. Operator manipulates the right joystick in the up direction.
2.5.14. Robot base of head moves in the positive pitch direction (up).
2.5.15. Operator manipulates the right joystick in the down direction.
2.5.16. Robot base of head moves in the negative pitch direction (down).
2.5.17. Operator manipulates the right controller trigger.
2.5.18. Robot base of head moves in the negative roll direction (clockwise).
2.5.19. Operator manipulates the left controller trigger.
2.5.20. Robot base of head moves in the positive roll direction (counterclockwise).

2.6. Alternative flow events

2.6.1. Robot reaches maximum rotation of a freedom of axis.
2.6.2. Operator input to move pass said limit is ignored.

2.7. Post Conditions

2.7.1. Robot joint moves in the intended direction unless pass limits.

3. Toggle AI Mode

3.1. User

3.1.1. Operator

3.2. Purpose

3.2.1. Operator wants to toggle AI mode in order for the robot to interact automatically with the public.

3.3. Preconditions

3.3.1. Robot is in manual mode.

3.4. Triggers

3.4.1. Operator's intent for the robot to interact with the public.

3.5. Flow of Events

3.5.1. Operator hits a button on the controller.

3.6. Post Conditions

3.6.1. Robot is in AI mode.

3.6.2. Robot automatically begins to run AI control routines.

4. Seeking Behaviour

4.1. User

4.1.1. Robot

4.2. Purpose

4.2.1. Robot wants to engage with persons not yet close to the robot in order to engage with interaction with them

4.3. Preconditions

4.3.1. Robot is in AI mode.

4.3.2. Robot is not currently interacting with a person.

4.4. Triggers

4.4.1. Robot is not currently interacting with a person.

4.5. Flow of Events

4.5.1. Robot moves head around to search for a person in its FOV.

4.5.2. If the robot hears sound, robot moves head toward direction of sound.

4.5.3. Robot detects a person at a distance greater than interaction distance but within seeking distance.

4.5.4. Robot is fixated on the User.

4.5.5. Robot moves head to point at said person.

4.5.6. Robot moves head in an up and down manner and makes goat noises.

4.5.7. Person moves within interaction distance

4.6. Alternate Flow of Events.

4.6.1. If fixated person leaves seeking distance or 10 seconds pass without moving into interaction distance, restart the flow of events.

4.7. Post Conditions

4.7.1. Person is now within interaction distance or no persons are detected inside seeking distance.

5. Non-physical Interaction

5.1. User

5.1.1. User

5.2. Purpose

5.2.1. User wants to test if the robot will respond to external movements in order to learn more about the robot.

5.3. Preconditions

5.3.1. Robot is in AI mode.

5.3.2. Robot is fixated on the User.

5.4. Triggers

5.4.1. User moves inside the interaction range while the robot is fixated on the User.

5.5. Flow of Events

5.5.1. User moves hand in a large wave motion.

5.5.2. Robot head and neck follows the movement of the hand.

5.5.3. User moves head in some movement.

5.5.4. Robot head and neck tracks user head.

5.5.5. User walks to another location still in robot FOV.

5.5.6. Robot moves head and neck so that User is in the center of FOV.

5.5.7. User reaches out hand to robot.

5.5.8. Robot moves head to touch hand.

5.6. Alternative Flow of Events

5.6.1. If User leaves interaction distance, robot removes fixation with the user.

5.7. Alternative Flow of Events

5.7.1. If User remain inside interaction distance but doesn't reach out hand, operator can guide user to do so.

5.7.2. If User remain inside interaction distance but doesn't reach out hand after 5 seconds, robot loses fixation.

5.8. Alternative Flow of Events

5.8.1. If another person reaches out hand, robot fixates on the new person.

5.9. Post Conditions

5.9.1. User and Robot are now in physical contact.

6. Physical Interaction

6.1. User

6.1.1. User

6.2. Purpose

6.2.1. User wants to touch the robot in order to feel the external texture and see how it responds.

6.3. Preconditions

6.3.1. Robot is in AI mode.

6.4. Triggers

6.4.1. User is touching the robot.

6.5. Flow of Events

6.5.1. User is touching head of robot.

6.5.2. Robot slowly moves head in a nodding motion while slowly rotating the base of neck along the roll axis (clockwise and counterclockwise). Robot also angles the head toward the direction of the touch (right, left, and up).

6.5.3. User is touching the neck of the robot from side of robot.

6.5.4. Robot moves head in the direction of user and moves head in an up and down motion.

6.5.5. User steps back and breaks physical contact.

6.6. Alternative Flow of Events

6.6.1. If User leaves interaction distance, robot removes fixation with the user.

6.7. Alternative Flow of Events

6.7.1. If User remains inside interaction distance but stops physical contact, robot removes fixation with the user.

6.8. Post Conditions

6.8.1. User is no longer fixated.

7. Sound Reactions - Otherwise

7.1. User

7.1.1. Robot

7.2. Purpose

7.2.1. Robot wants to engage with a person or group of persons through auditory ways.

7.3. Preconditions

7.3.1. Robot is in AI mode.

7.3.2. Robot is fixated on a person.

7.3.3. Persons are within interaction distance.

7.3.4. Sound is not being made in response to a conversation.

7.4. Triggers

7.4.1. Sound is detected.

7.5. Flow of Events

7.5.1. Sound is detected.

7.5.2. Robot makes a sound.

7.6. Post Conditions

7.6.1. Robot is still fixated on the same person.

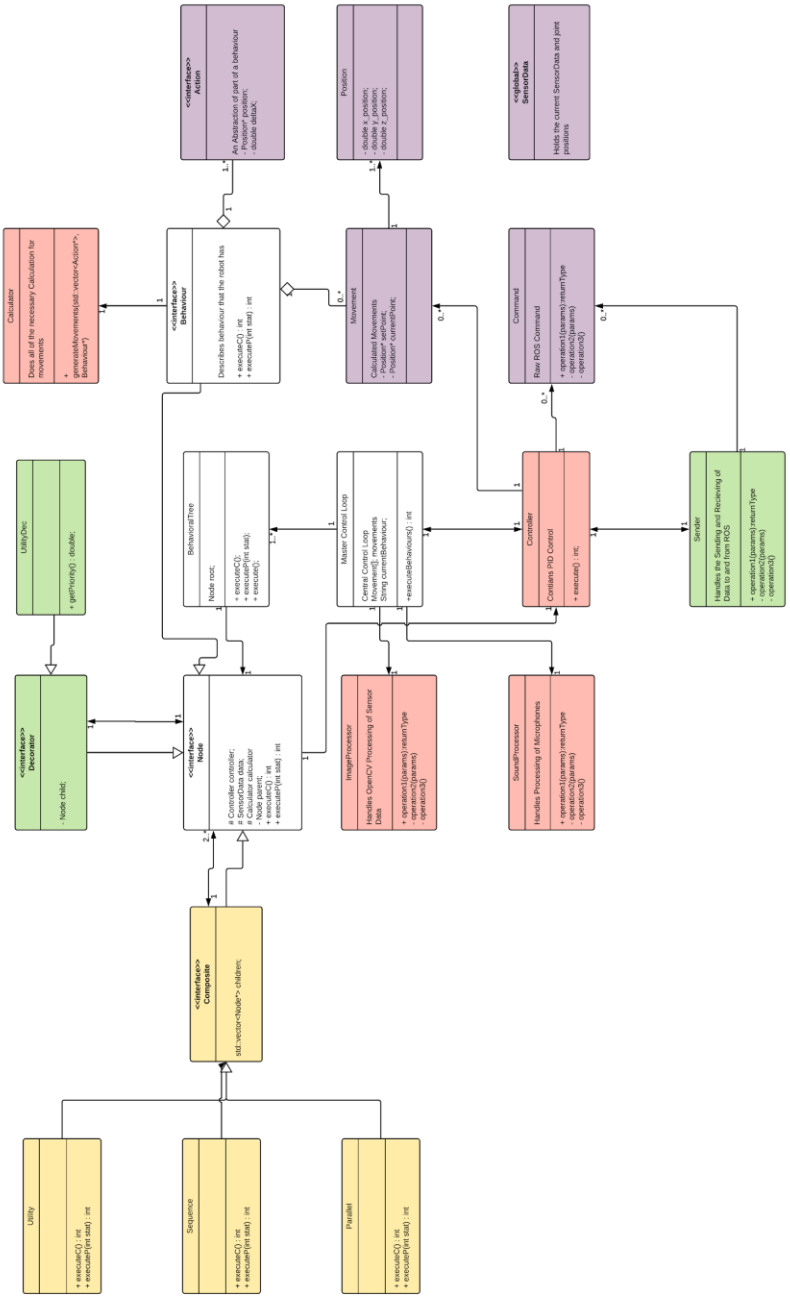
Appendix H: Prioritized Use Cases

1. Seeking Behaviour
    a. Recognition of humans on camera and movement to face and interact with them
    b. Seeking of humans on camera is prioritized as the highest since it is a steppingstone to the next highest behavior which is nonphysical interaction. Being able to detect humans allows the robot to do that behavior as well as seek out new persons. It also requires the AI to be able to calculate expected locations and be able to develop the necessary commands to move the robot to that position.
2. Non-Physical Within Interaction Distance
    a. Waving Hands, moving heads, tracking movement
    b. This nonphysical interaction is prioritized as the second because it relies on functionality from the first, but it is also probably going to be used the most by people interacting with the robot. This behavior also doesn't need to overcome the social barrier of touching robot and its behavior is movement that nearly everyone can trigger just be looking at the robot.
3. Physical Interaction
    a. Reacting to physical touch, limiting movement based on touch sensors
    b. This is likely to be the next most used behavior as once one person sees the robot being touched, those more extroverted might be willing to touch the robot. These movements will also be very important in selling the believability of the lifelike goat that all this behavior is driving toward.
4. Sound Detection Response
    a. Detect sound and make a noise in response
    b. This is a behavior that everyone might not use although some persons might say hi; it is unlikely that everyone will. Furthermore, this step is

relatively simple compared to the previous behavior and therefore can be delayed.

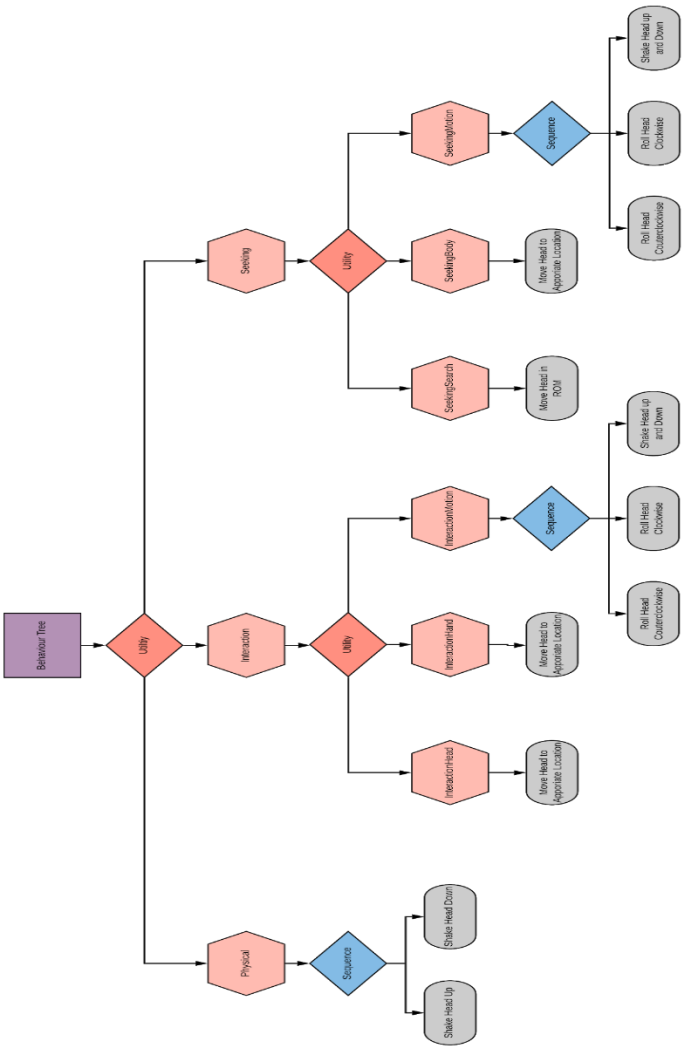
5. Conversation Detection
    a. Robot able to detect differences between conservation and directed cues
    b. This particular behavior is important for the robot to not interrupt conversations and is the next step after sound detection. It would allow people to have conversation near the robot without interruption which sells the goat being able to only react to sound direct toward it, a more lifelike behavior.
6. Sound Location Detection
    a. Detect a sound and have the robot face direction of sound and fixate on person who made sound
    b. This is another behavior that will probably be rare as it only occurs if the robot has not yet fixated on a person and the person trying to get its attention is outside its FOV. Although a nice feature that definitely helps the believability of the robot it is not likely to be used enough to make it a higher priority.
7. Hand Nuzzle Behaviour
    a. Robot goes out of outstretched hand
    b. This behavior is low priority because of the difficulty of doing it and low likelihood of people outstretching their hands. Guiding the head of the robot to a hand in its visions is far harder than just pointed the head toward that hand. Furthermore, it was determined that a person is far more likely to just touch the head than top reach out a hand.

Appendix I: Object Diagram



[Click Here to See the Full-Size Object Diagram]

Appendix J: Behavior Tree Diagram



[Click Here to See the Full Size Behaviour Tree]

Appendix K: Decorator Design

1. Physical Utility Decorator
    a. Max 1.0
    b. Min 0.0
    c. If the touch sensor is touched? 1.0 : 0.0
2. Non-physical utility decorator
    a. Max 0.9
    b. Min 0.0
    c. Ratio of Person in robot FOV
        i. If person has been fixated multiplier of 0.5
3. Seeking Utility Decorator
    a. Max 0.3
    b. Min 0.3
    c. Steady Minimum
4. Time Since Last Sound Decorator
    a. Max 1.0
    b. Min 0.0
    c. Starts at 1.0, degrades to 0.0, after period returns to 1
5. Person Hand is Moving Decorator
    a. Max: 1.0
    b. Min 0.0
    c. If Robot Does Not See Hand
        i. 0.0
    d. If hand in within bounding box in the center of FOV
        i. 1.0
        ii. Largest Hand Takes Priority
6. Person Head Not in Center of FOV
    a. Max 0.9
    b. Min 0.0
    c. If Robot Does Not See Head

i. 0.0

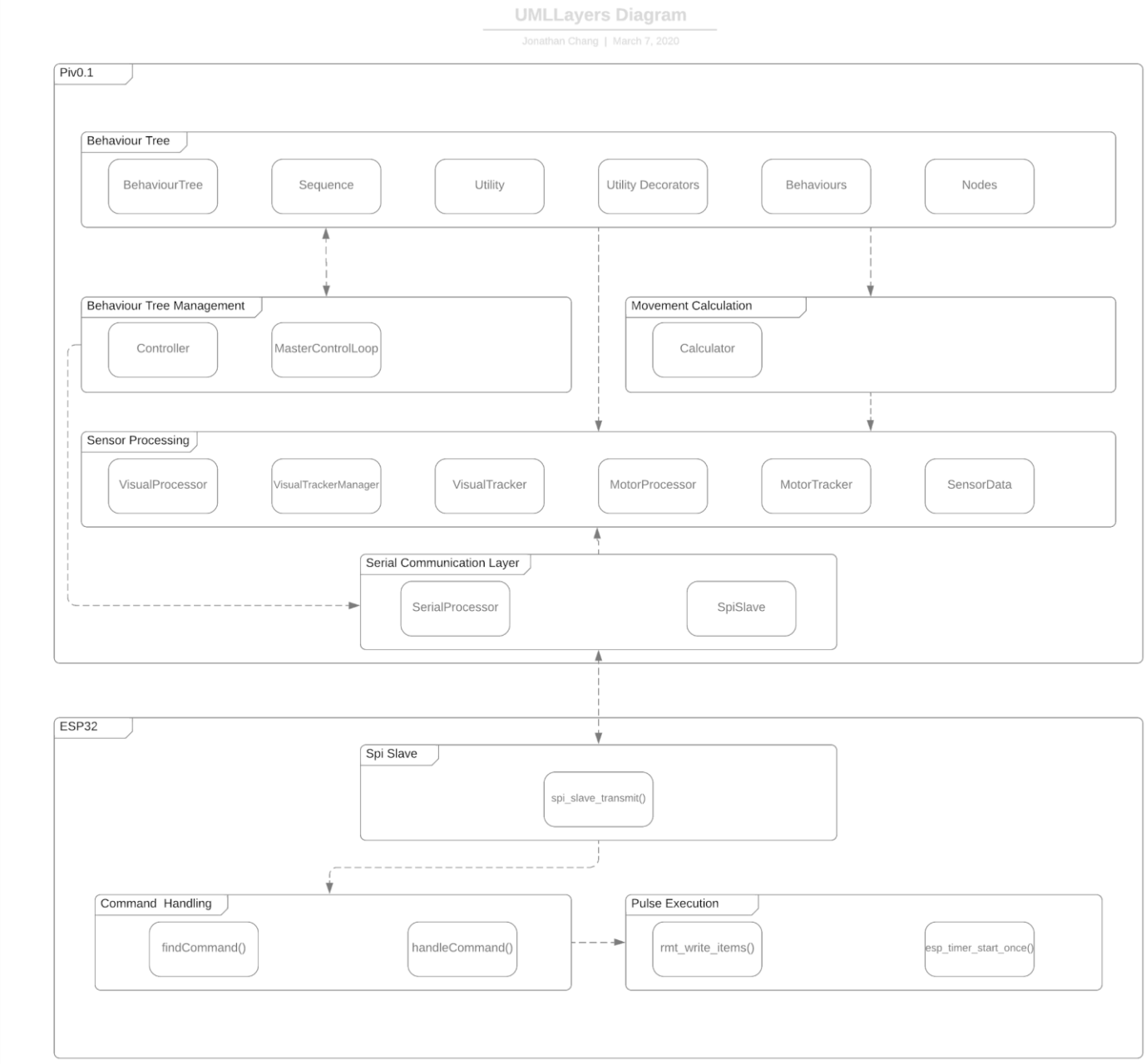
d. If head in within bounding box in the center of FOV

i. 0.9

ii. Largest head Takes Priority

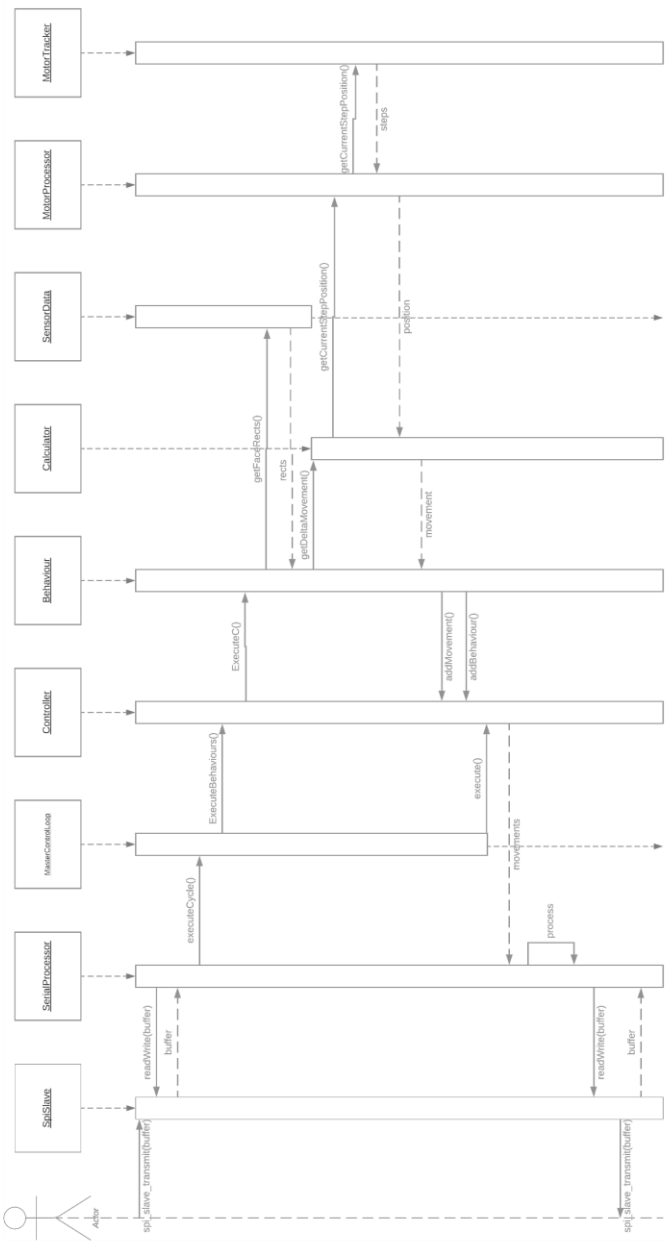
7. Person Motionless Decorator
    a. Max 0.3
    b. Min 0.3
    c. Steady Minimum
8. Seeking Head Decorator
    a. Max 0.3
    b. Min 0.3
    c. Steady Minimum
9. Person in View to Seek Decorator
    a. Person Hand is Moving Decorator
        i. Max: 1.0
        ii. Min 0.0
        iii. If Robot Does Not See Hand
            1. 0.0
        iv. Else
            1. 1.0
        v. Largest hand Takes Priority

Appendix L: Layer Diagram



[Click Here to See a Full-Size Layer Diagram]

Appendix M: Sequence Diagram



[Click Here to See the Full-Size Sequence Diagram]

Appendix N: Future Software Work

1. UUID of Behaviours
    a. UUID is given on POSITIONCMD and are unique to a single behavior execution not a behavior. REQUESTCMD returns UUID of completed behavior when all movements are completed for that UUID. This would allow for parallel behavior in the behavior tree
2. Fix Motor Ratios
    a. In the calculator, the ratio of movement between the head and neck is calculated as a decimal but it should take into account the maximum movements in the head to adjust
3. Behaviour Tree Execute Cycles
    a. Make each execute cycle of the behavior tree depending on serial communication of REQUESTCMD instead of CMD line
4. CommandLineProcessor
    a. A SensorProcessor object that manages all user inputs through the command line. There could be manual commands to start, stop, or manually move the robot here.
5. AudioProcessor
    a. Actually, process noise data and make sound as necessary. This is part of the required behavior.
6. Additional Commands
    a. Having commands to handle enable motors, start or stop sensor information.
7. Less reliance on globals
    a. Store globals as pointers and set the pointers on startup. This will also make testing easier.
8. Testing
    a. Unit Tests for all Objects
9. SensorData

a. Make behaviours and decorators use SensorData as an interface to access processor data. This is better style even if it adds an additional layer.

Appendix O: Code Base

The Entire Code Base is available on Github:

ESP32 Code: https://github.com/WPIMascotMQP/ESP_32-SPI/tree/stepping

Raspberry Pi Code: https://github.com/WPIMascotMQP/version0.1/tree/piv0.1

General Testing and Prototyping Code :

https://github.com/orgs/WPIMascotMQP/teams/members/repositories