Advancing Humanoid Robots: Demonstration of Standing and Assisted Walking Alongside a New Simulation Framework

IN PARTIAL FULFILLMENT OF A MAJOR QUALIFYING PROJECT

WORCESTER POLYTECHNIC INSTITUTE

Submitted by:

Stephen Fanning (RBE/CS)

Ana Roure (RBE)

Dylan Nguyen (RBE/CS)

Jack Rothenberg (RBE/BME)

Jatin Kohli (RBE/CS)

Scott Pena (RBE)

Project Advisors:

Prof. Pradeep Radhakrishnan (MME)

Prof. Dirk Albrecht (BME)

Prof. Kaveh Pahlavan (CS)

Abstract

This project is a continuation of the multi-year 3D-printed humanoid robotics MQP aiming to iterate on the simulation, electrical, and firmware solutions of previous years to create a robust, reliable, and efficient open-source humanoid robot. Ava is the current iteration of this robot. Her electrical system was updated to remove points of failure commonly found in moving wiring solutions through the development of a central circuit board and custom wires. The simulation, in CoppeliaSim, was updated to include a more accurate model of the robot along with material and mechanical properties. In simulation, Ava is now capable of unassisted standing for over 3.5 hours, squatting for over an hour, and walking with a cart for over an hour. In real life, Ava is now capable of unassisted standing for 3 minutes and walking with a cart for over 2 meters. Ava utilizes high-level software through a Raspberry Pi 4B and low-level firmware through an Arduino MEGA 2560 to interact with the world. Using a 9-DOF BNO055 IMU and trajectory planning with kinematics, she demonstrates walking forward while pushing a cart and standing upright independently.

Acknowledgments

This project would not have been possible without extensive help from the following people:

Prof. Pradeep Radhakrishnan
Mr. Peter Hefti
Prof. Dirk Albrecht
Prof. Kaveh Pahlavan
Emily Austin (Graduate Student)
James Van Milligen (Graduate Student)
Ms. Barbara Furhman
Mr. Cam Tu Le
The 2023-2024 3D Printed Humanoid Robot MQP Design Team

Table Of Contents

Nomenclature

Symbol/Variable	Meaning
x_{ZMP}, y_{ZMP}	x and y coordinate of the zero moment point
m	mass
p_{ix}, p_{iy}, p_{iz}	coordinates for the center of mass of a link in a manipulator
$\ddot{p}_{ix}, \ddot{p}_{iy}, \ddot{p}_{iz}$	acceleration of the center of mass of a link in a manipulator
g_z	gravity vector
I_{ix}, I_{iy}	x and y parts of an inertia tensor
F_{RN}	resultant normal force due to the sum of all normal forces exerted on the bottom of Ava's foot.
F_N	normal force exerted on Ava's foot
p_{CoP}	position vector of the center of pressure
p_{FNi}	position of the normal force
S	spatial twist
ω, v	angular and linear components to a spatial twist
R_ω	rotation matrix of a rigid body
$e^{[\omega]\theta}$	exponential coordinates representing the rotation of a rigid body
$e^{[S]\theta}$	exponential coordinates representing the rotation and translation of a rigid body
$T(\theta)$	transformation matrix for kinematic calculations

M	transformation matrix representing the manipulator's home configuration
$J(\theta), J_n$	jacobian matrix, each column in a jacobian matrix
$V, \dot{\theta}$	linear and angular velocity of a manipulator's end-effector
Ad_n	adjoint transformation matrix

Abbreviations

Abbreviation	Meaning
WPI	Worcester Polytechnic Institute
MQP	Major Qualifying Project
0101	HerkuleX DRS-0101
0201	HerkuleX DRS-0201
0601	HerkuleX DRS-0601
UART	Universal Asynchronous Receiver/Transmitter
CoM	Center of Mass
CoP	Center of Pressure
ZMP	Zero Moment Point
IMU	Inertial Measurement Unit
VS Code	Visual Studio Code
GPIO	General-Purpose Input/Output
IDE	Integrated Development Environment
SSH	Secure Shell Protocol
VNC	Virtual Network Computing
PID	Proportional Integral Derivative
JST-XHP	Japanese Solderless Terminal Plug Housing

CAD	Computer-Aided Design
LiPo	Lithium-ion Polymer
FDM	Fused Deposition Modeling
URDF	Unified Robotics Description Format
API	Application Programming Interface

Authorship Table

Section	Author(s)
Abstract	Jatin & Jack
Introduction	Jatin & Ana
2021-2022 MQP	Ana & Stephen
2022-2023 MQP	Ana & Stephen
Goals	Jatin & Ana
Literature Review	Scott & Jack
Methodology	Ana, Stephen & Jack
Robot Evaluation	Ana & Dylan
Circuitry	Dylan & Stephen
Kinematics	Scott & Jack
Trajectory Planning	Scott & Jack
Simulation	Scott & Jack
Locomotion Software	Jatin & Dylan
Results	Stephen, Ana, Jack, Dylan, Scott & Jatin
Discussion	Jack & Jatin
Conclusions	Scott, Jack & Dylan

List of Figures

List of Tables

1. Introduction

In 2012, an open-source 3D-printed humanoid robot called Poppy was created by the Flowers laboratory at the University of Bordeaux [1]. Starting as a PhD thesis, the project aimed to provide a community-developed design for a humanoid robot to make the platform more accessible to all for development and research. Anyone can contribute to the project or create new projects based on the work of the Poppy project. Poppy is a toddler-sized humanoid robot with the capability of doing human movements.

In the 2021-22 academic year building off of the Poppy project, a team working on a Major Qualifying Project (MQP) at Worcester Polytechnic Institute (WPI), designed a 3D-printed humanoid robot named Koalby [2]. They recognized the Poppy project while being fully open source. The cost of recreating Poppy was really high thus their main goal was to modify the Poppy robot to be more affordable. Thus, they created Koalby, a much cheaper robot compared to the Poppy robot and with a new purpose to be a lab assistant robot.

In the 2022-23 academic term, another team of WPI students continued the work of the 2021-2022 team. Their main goal was to enhance Koalby's availability to work as a laboratory assistant, specifically its ability to walk [3]. They did this by adding sensors like cameras and IMU sensors. They also took the task of creating an extra robot with a new gripper design with the capability of grabbing objects. The new robot was named Ava.

This academic year (2023 - 2024), we continued the work of these past teams to develop Koalby and Ava into functional robots to work as laboratory assistants and at-home assistants. As the locomotion team, we are working with this year's design team to iterate on the previous year's work. Specifically, we aim to make the robots able to reliably perform basic gestures, stand unassisted, and walk with assistance for the first time in their history. With these aspects, we are hoping that Ava and Koalby will one day compete in RoboCup.

1.1 Report Organization

Our report is separated into several chapters based on different areas of our research and work. Chapter 2 outlines the foundational work on Koalby by the original 3D-printed humanoid robot MQP teams in 2021-2022. Chapter 3 summarizes the previous year's iteration of Koalby as well as the creation of Ava during the 2022-2023 3D-printed humanoid robot MQPs. Chapter 4 defines our goals and objectives for this year's locomotion software. Chapter 5 contains our literature review regarding control algorithms for humanoid robotics. Chapter 6 lays out our methodology used to approach the multi-faceted problems tackled during each of our four terms of work. Chapter 7 presents work completed in collaboration with the 2023-2024 printed humanoid robot design MQP team to replace and re-design inadequate 3D printed parts on Ava. Chapter 8 describes our custom circuitry solution crafted as a result of many rounds of iteration and testing. Chapter 9 explains our approach to calculating Ava's forward, inverse, and differential kinematics for use in motion planning. Chapter 10 details our trajectory planning methods used for calculating paths for Ava's hands and feet. Chapter 11 illustrates our overhaul of last year's simulation environment and implementation of several locomotion capabilities including standing, walking, and squatting. Chapter 12 talks about our development and testing of the software running on the physical robot's Raspberry Pi 4B and Arduino MEGA 2560. Chapter 13 introduces the results and outcomes of our development, integration, and testing of multiple subsystems into a robot capable of advanced locomotion capabilities. Chapter 14 discusses our results as compared to our goals and methodology and also lays out recommendations for future teams. Chapter 15 recaps our work throughout this MQP in our conclusion.

2. 2021-22 MQP

The inaugural team assigned to this project was the 2021-22 MQP team [2]. They undertook the task of designing a robot inspired by the Poppy project, incorporating modifications aimed at cost reduction and power enhancement. The robot they created was named Koalby and by the end of their project, they were able to have Koalby dance with the help of a frame.

2.1 Poppy and Koalby

The Poppy Project is an open-source humanoid robot initiative, originating from the Flowers Laboratory, with the primary objective of advancing and popularizing the adoption of cost-effective humanoid robots in research and education [1]. These robots are designed to be fully 3D printed and have around 25 degrees of freedom distributed across various joints. The Poppy robot was made using DYNAMIXEL motors, which are visible in Figure 2.1 [4].

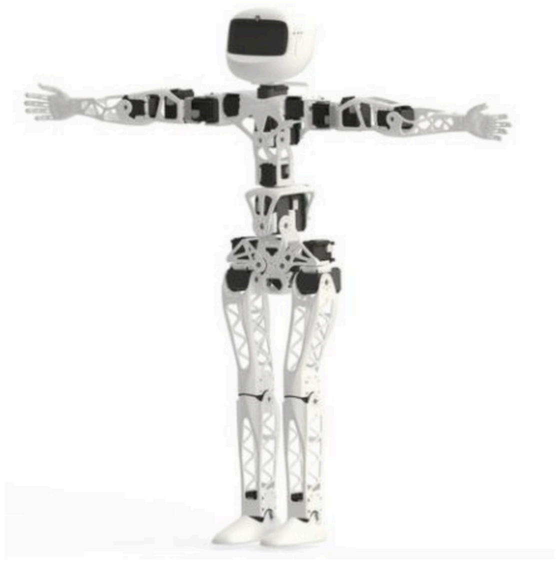

Figure 2.1: Poppy Humanoid 1.0 (Reproduced as is from [1])

2.1.1 Koalby

Koalby, seen in Figure 2.2, is the robot the 2021-2022 team created inspired by the Poppy project. Koalby is toddler-size and fully 3D printed like the Poppy Robot with a couple of modifications to reduce the cost of the robot significantly. The main changes were adding LiPO batteries to the legs and replacing some of the DYNAMIXEL motors with HerkuleX motors.

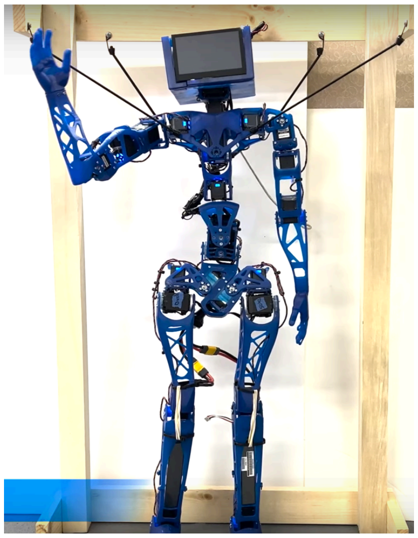
Figure 2.2: Koalby Waving (Reproduced as is from [2])

2.1.2 Koalby Motors and Batteries

As previously mentioned, the team implemented several cost-saving modifications to the robot. One of the most notable changes involved replacing some of the DYNAMIXEL motors used in Poppy with HerkuleX motors, which offer similar performance at a significantly lower cost. DYNAMIXEL motors were priced, in 2021/22, at \$260.00 with a stall torque of 2.50 Nm, whereas HerkuleX motors are available for \$132.00 with a stall torque of 2.35 Nm. Currently, the previously used DYNAMIXEL motors are priced at \$299.90 and their HerkuleX replacements are priced at \$154.00. These changes reduced the total cost of motors by approximately \$2500, as illustrated in Table 2.1 with the respective comparisons.

Table 2.1: Comparing DYNAMIXEL MX-28 and HerkuleX DRS-0201 Motors (Reproduced as is from [3])

Motor	**Cost**	**Stall Torque (Nm)**	**No load speed (RPM)**	**Form Factor (mm)**
Dynamixel MX-28	$260	2.5	55	32 x 50 x 40
HerkuleX DRS-0201	$132	2.35	68	24.0 x 45 x 31

The second major alteration centered around the power source. While Poppy relied on a wall outlet for power, Koalby is now powered by three onboard batteries. Two of the LiPO batteries are 7.4V and are placed in parallel on both shins of the robot, as seen in Figure 2.3, and one was 11.1V and was placed in the head.

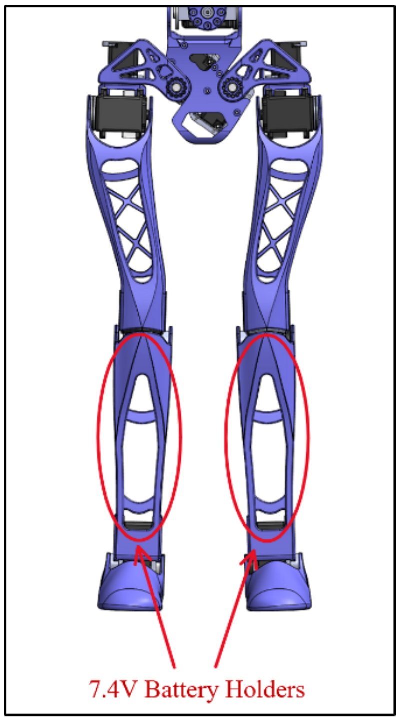

Figure 2.3: 7.4V Battery locations in Koalby's Legs (Reproduced as is from [3])

2.1.3 Software

The code structure for Koalby was written in both Python for a Raspberry Pi 4B and Arduino for an Arduino MEGA 2560. Their Arduino was responsible for controlling the robot, specifically managing the motor movements and sensor data. In their code, they had two main sections: the setup where it would initialize serial communications with Python, and the loop which continuously executes specific methods/commands within the robot class for motor control and sensor data retrieval. The list of commands can be seen in Table 2.2.

Table 2.2: Koalby Commands (Reproduced as is from [3])

Command	**Number**	**Additional Parameters**	**Description**
Init	1	0	Initialize all motors, move them to home positions
GetPosition	5	1	Return position of the motor, normalize to 0-100 range
SetPosition	10	2	Set position of the motor to a given value, normalize to 0-100 range
SetPositionT	11	3	Move the motor to a given value in 0-100 range, take the specified amount of time to travel there
ArmMirror	15	1	Right arm is disabled, left arm moves to a position based on where the user moves the robot’s right arm. This is primarily a proof of concept and will be housed on the Pi in the final version
SetTorque	20	2	Set the torque of a motor to either on or off

SetCompliant	21	2	Sets the motor to either normal or compliant mode
Shutdown	100	0	Disable all motors

The Python code was used to have all of the functions for the robot to move. They were able to code Koalby making various movements like dancing, shaking hands, and waving. For these functions to work they used a file called “ReplayPrimitive,” which would read all of the motor positions which would then set all of the robot's motor angles. This is then used to send this information to the Arduino for the Arduino to then send the information to the actual motors.

This chapter described the journey of the 2021-2022 MQP team developing Koalby, a toddler-sized robot inspired by the Poppy initiative. By integrating cost-saving modifications like substituting some DYNAMIXEL motors with more affordable HerkuleX motors, they reduced the total motor cost by approximately $2500. The next chapter will discuss the second team working on the continuation of the project.

3. 2022-23 MQP

The 2022-23 MQP team constituted the second phase of this MQP [3][7]. Their goals focused on enhancing Koalby's ability to perform at-home tasks and also undertook the construction of an additional robot named Ava with improvements to the design. These modifications significantly improved the physical attributes of Koalby, notably enhancing its capability to stand independently. Furthermore, with the creation of the extra robot Ava, the design team was able to make modifications to the robot to improve its capability to pick up and place objects with a new gripper design and integration of different sensors.

3.1 Development of Ava

The 2022-23 design MQP team was tasked to create Ava, an open-source 3D-printed humanoid robot that would act as a lab assistant with a focus on lifting objects and assisted walking via pushing a cart [7]. Their goal was to implement design changes to Koalby that help the robot with structural integrity, walking, standardizing components, and gripping objects.

To create Ava, they improved the structural integrity by redesigning Koalby's parts to improve the strength of Ava. They gave Ava walking capabilities by integrating electrical components like adding an IMU, Arduino, and Raspberry Pi in Ava. In their final design seen in Figure 3.1, they added gripping functionality to lift objects and help grab metallic objects by adding an electromagnet into its hand.

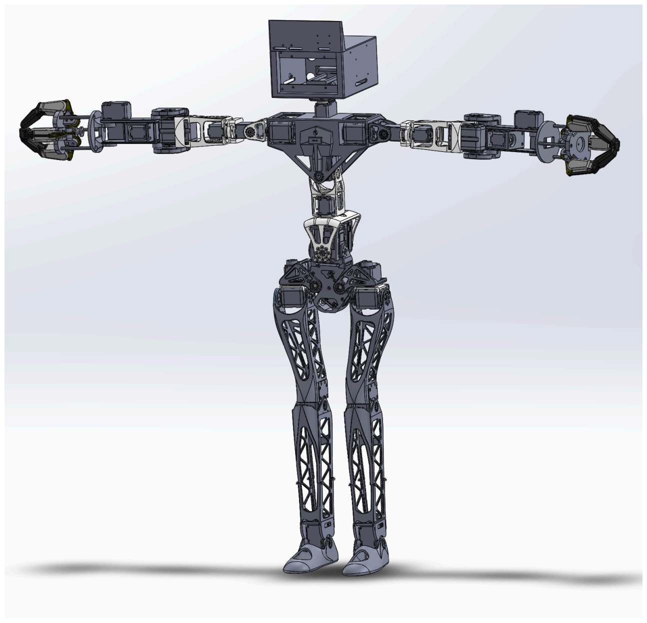

Figure 3.1: Final 3D model for Ava (Reproduced as is from [7])

3.2 2022-2023 Controls/Simulation Team MQP

The 2022-2023 Controls MQP team was tasked with continuing the work initiated by the 2021-2022 controls team [3]. Their overarching goal was to enable the robot to walk with the assistance of a cart and manipulate objects by picking them up and placing them. With this objective in mind, the team outlined smaller goals to facilitate assisted walking. These included assessing and improving the physical robot by incorporating more sensors into Koalby's functionality, upgrading Koalby's motors to have only HerkuleX, recreating the robot in simulation to ease testing, and developing an interface that would allow users outside the development team to meaningfully interact with Koalby.

3.2.1 Motor and Sensor Integration

As mentioned previously, Koalby had both HerkuleX and Dynamixel motors and both motors are extremely similar to each other. For this project, the HerkuleX motor's resolution, operating angle, and speed met the requirements to perform the walking motion. To be more specific, the HerkuleX motors gave enough torque to be able to perform walking motions from moving the leg up and carrying the weight of the robot. Thus the HerkuleX DRS-0601 and

DRS-0101 can replace the Dynamixel MX-64AT and AX-12 motors, respectively. With this in mind, the team decided to fully replace all of the Dynamixel motors with HerkuleX motors. Koalby has 19 HerkuleX DRS-0201 motors, eight HerkuleX 0601 motors, and two HerkuleX DRS-0101 motors. The DRS-0201 motors are used for joints like the arms, elbows, and knees. The HerkuleX DRS-0601 motors are placed at joints expected to require high torque, like the shoulders and thighs. You can see where the motors were placed in Figure 3.2.

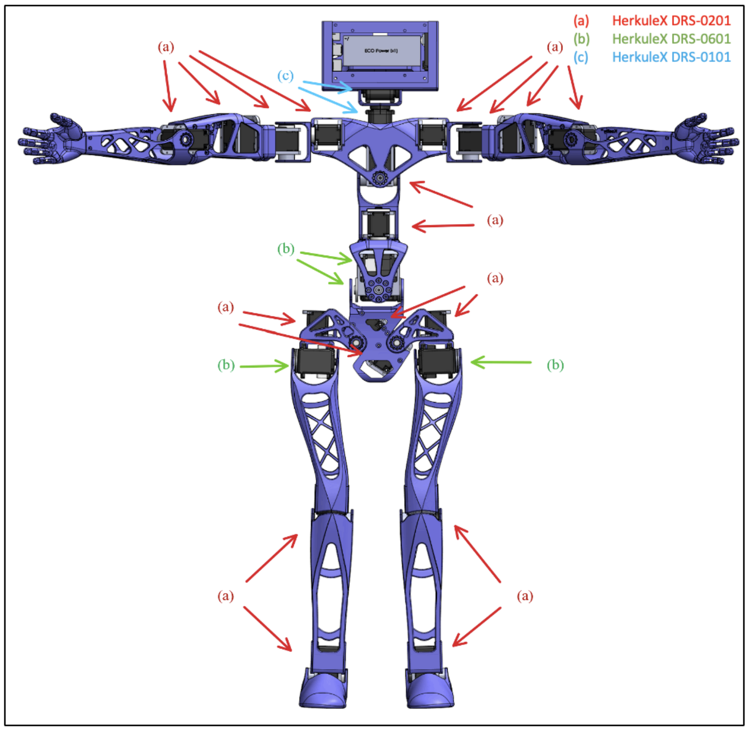

Figure 3.2: Locations of HerkuleX Motors in Koalby (Reproduced as is from [3])

With the help of the design team, they integrated sensors like an IMU, Huskylens, and TF Luna. The Adafruit BNO055 IMU is a 9-DOF IMU that was used for stability detection which they placed to the lower part of the torso which, according to the team, was close to the center of mass. The AI Camera Huskylens was used for object and proximity detection and was placed on top of Koalby's head. Finally, like the Huskylens, the TF Luna LiDAR sensor was also chosen for proximity detection and was placed on the head of the robot to detect objects while walking.

3.2.2 Circuitry

Their circuitry consisted of a 5-way split, with solid core wires placed in the back of Koalby and around the motors, daisy-chained to each other. New motors meant new wiring diagrams and new circuitry. They added two 11.1V batteries to provide power to the HerkuleX DRS-0601. Voltage regulators are used to output 7.4V, which then powers the HerkuleX DRS-0201 and 0101s. You can see their wiring diagram in Figure 3.3.

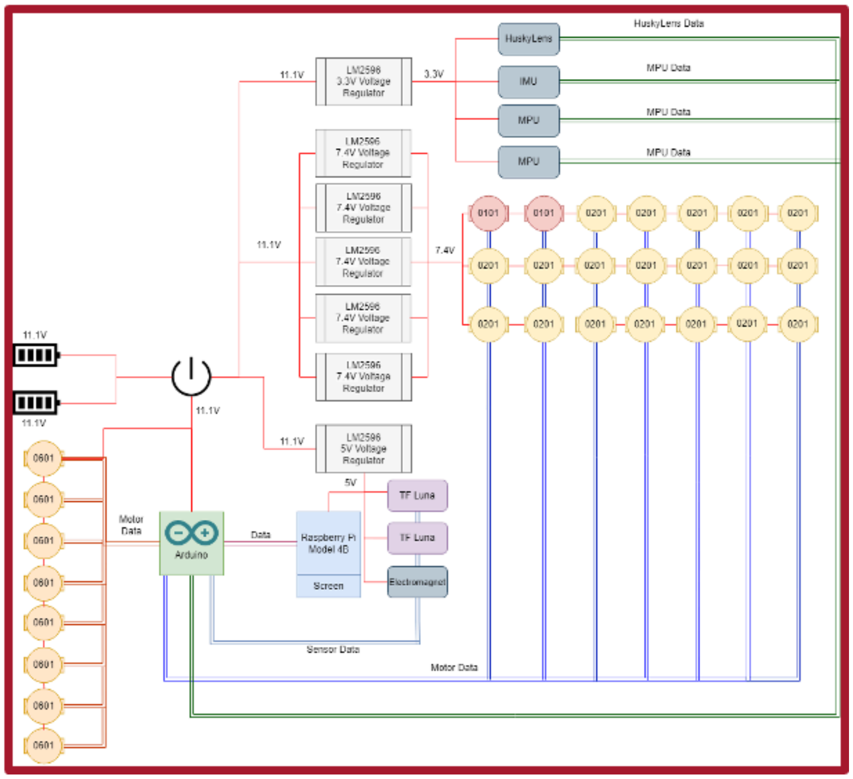

Figure 3.3: Final electrical diagram (Reproduced as is from [3])

3.2.3 Simulation

Last year's controls team was the first to perform work towards making a simulation for the robot. They considered several different software including Blender, Gazebo, and CoppeliaSim [3]. They decided against using Blender because it was too demanding on the computer to run, and it did not have many features that would allow for the streamlining of work in robotics including sensors. They decided not to use Gazebo, either. Although it is designed to work with robots and has many useful plug-ins to allow for improved workflow, the team preferred to utilize computers with Windows or Mac operating systems instead of Linux. They chose CoppeliaSim due to it being the best of the three they looked into, but also because it was easy to construct the robot as they wanted in the software, control it using Python and data from integrated sensors, and is what the existing Poppy robot currently uses.

Once deciding on using CoppeliaSim, they created a simplified, block model of the robot inside CoppeliaSim using its native construction tools as seen in Figure 3.4. They initially tried to import URDFs provided by the 2021-2022 team but were unable to resolve the various errors and bugs they encountered. The simplified simulation model was missing many details from the real world, such as accurate 3D geometry, mass and inertia information, and joint torques. Additionally, a joint in each arm had an inaccurate orientation, thus rotating about the wrong axis, as well as each arm was missing a joint to represent the real robot's wrist.

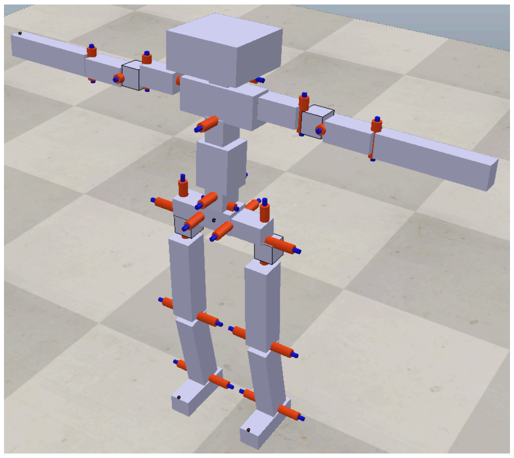
Figure 3.4: Screenshot of Last Year's Model in CoppeliaSim [3]

Inside the simulation, the team was able to achieve basic control of each joint using an internal PID controller within CoppeliaSim. They were able to set the target angle of a specific joint in a Python script and then move the joint in CoppeliaSim by using a remote API call to set the joint's target position. Additionally, the team calculated the kinematic information of the robot's arms and legs to calculate forward and inverse kinematics. They primarily used this data to create cubic trajectories for the legs to simulate a walking motion.

Lastly, the team also built another scene in which the robot was holding onto a cart. They accomplished this by attaching one arm to the cart with a fixed joint. Using this setup, the team was able to achieve assisted standing for an indefinite amount of time.

In the simulation, using the built-in PID controller of CoppeliaSim with the IMU readings that they recorded at their most stable unassisted standing pose, they were able to have the robot stand unassisted for over five minutes. After investigating their code and running the simulations they left us, we were unable to recreate this result, with the robot falling after a few seconds.

In addition to standing unassisted, they also simulated the robot standing with the assistance of a wheeled cart. They used the same built-in PID controller as with the unassisted stand, but they did not report a duration they were able to attain. Unlike with unassisted standing, they were able to get the robot to stand assisted indefinitely.

3.2.4 Kinematics and Trajectory Planning

Last year's team also calculated the kinematics regarding each of Koalby's limbs to increase the control of his hands and feet. The forward kinematics of each arm and leg were calculated via Denavit-Hartenberg parameters, which were hardcoded into the code [3]. This allowed the team to calculate the location and orientation of the limb's end effector (the hand or foot) when given a set of joint angles.

To calculate the inverse kinematics of the leg, this team used the geometric method. When provided desired cartesian coordinates of the end-effector, the inverse kinematics were solved within the software. For the inverse kinematics of the arms, the team tried using twists for calculations, but no implementations were created in simulation or on the real robot.

The team also implemented cubic trajectory planning for limb motion. They decided to use a cubic trajectory to ensure that the end-effector's motion had a constant velocity. They used their trajectory generator to reenact various primitive movements from the previous year's team, but also to begin testing assisted walking. They demonstrated a basic first step using cubic trajectories but did not successfully implement a walking gait.

The 2022-23 MQP team upgraded Koalby's household capabilities and introduced Ava, a humanoid robot designed for object manipulation and cart-assisted walking. The next chapter will discuss the goals and requirements for our team which is the next and third team for this project.

4. Goals and Requirements

Our primary objective for the project was to enable Ava to walk with the assistance of a cart by presentation day. Recognizing the current non-functional state of Ava and Koalby, as well as our ambitious long-term aspirations of competing in RoboCup @ Home, we aimed to establish the foundational framework of a walking, gesturing humanoid robot. Our focus prioritized locomotion over auxiliary sensors such as TF Luna and Husky Lens, which last year's team had explored. To achieve unassisted walking and standing gestures, we outlined intermediary goals to accomplish by the end of each term. These goals were categorized into key areas: the physical robot, circuitry, simulation, and software. Progress across these categories was addressed throughout each term, culminating in the realization of our primary goal for presentation day.

4.1 List of Requirements

For each of our major goals, we outlined specific numerical requirements for the robot to fulfill. These are the numbers we used to determine whether our project had succeeded in its individual goals:

- Unassisted standing for at least 5 minutes
- Assisted walking for at least 1 meter
- No circuitry failures within a 30-minute demonstration

These served as our primary objectives throughout the year, and we broke these down into smaller tasks to be completed in each term.

4.2 Timeline of Activities

We began each term by outlining a list of goals and requirements for that term specifically, as well as broader goals for future terms. The resulting timeline of our goals broken down by term is described in the following subsections.

4.1.1 A Term

For A term, we aimed to get Ava and Koalby functional. This entails fixing all of the broken 3D printed parts and deploying code to both robots as well as demonstrating primitive movements (waving, dancing, etc.) with Ava. Being able to deploy code and send the robots to arbitrary joint configurations would allow us to begin work on unassisted standing and walking in the future. We also planned to experiment with CoppeliaSim to recreate the previous team's progress and determine its feasibility for simulating standing and walking for eventual implementation in the real world.

4.1.2 B Term

For B term, one of the main goals was to have the robots standing unassisted in simulation by implementing a dynamic model of the robots and PID control over the joints. We hoped this would help us get both robots to balance while standing unassisted in real life. We also planned on only focusing on working with Ava to have her standing unassisted. Standing unassisted is an important first step and proof of concept for walking unassisted as well as a major part of the final demo. Another primary goal was to update our circuitry plan for robots to use so they would not have electrical issues while moving. As secondary goals, we intended to develop a website to showcase our work, specifically to potential sponsors, to secure funding and document our journey. Our final goal was to refactor our codebase to make sense of last year's work.

4.1.3 C Term

For C term, our goal aimed to have the robots walking with assistance in simulation to assist with implementing walking functionality in the real world. We also wanted to finish implementing the new updated wiring plan for Ava that way when testing code on Ava no electrical issues happen and future teams don't need to work on the electrical side of the project and focus on the software side only.

4.1.4 D Term

For D term, we planned to combine all of our past work to create a final demo where Ava can stand unassisted and walk around with the help of a cart. We will create transfer documents for future teams so that they will not need to spend time figuring out where our team left off. We also planned to finalize deliverables for our simulation, dynamic modeling, and website to show up during the presentation as well.

4.2 Conclusion

These goals and requirements served to guide our overall project management and inform day-to-day decisions. Each task distributed to a team member was directly tied to one of these goals, ensuring that each of our tasks was actively contributing to the success of the project. Additionally, our numerical requirements allow us to prove the success of the project and ensure that we completed each of our major objectives.

5. Literature Review

Given that the goals of the project for this year were very strongly targeted towards overhauling both hardware and software systems to enable simulated and real motion, our literature review was strongly targeted towards practical articles that would act as references for us to look to for guidance while developing our systems. Because of this, there are two main categories we analyzed in the literature, those being two major decisions we had to make: what simulator to use and what kind of algorithms to use to control the robot.

5.1 Human Walking Gait

When humans walk, we move through what is known as a gait, which is how a person walks. A human goes through a gait cycle, where each of the start and the end of a cycle is the same repeatable action done by one limb [8].

Throughout a human's gait, each leg goes through a swing phase and a stance phase. A swing phase is when the foot is 'swinging' through the air and is not supported by the ground. On the other hand, the stance phase is when the foot is planted on the ground and supports the weight of the body. When both feet are planted on the ground during the gait, then the body is in what is known as a double support phase. If one leg is in a swing phase and the other leg is in its stance phase, then the body is in a single support phase, as the weight of the body is only being supported by one leg. As a human walks, their gait cycles through a double support phase and a single support phase, alternating which leg is in a swing or stance phase.

5.2 Kinematics

When actuating a limb on a humanoid robot, it is very important to have an understanding of how the position and angular velocity of each joint affect the end-effector's motion. Additionally, it is important to know what joint angles are needed to move the end-effector to a desired position and orientation in the robot's workspace. Therefore, kinematics calculations are used to define the relationship between a manipulator's joint space and its task space. There are numerous textbooks written on these subjects, but we primarily utilized Modern Robotics.

Modern Robotics is a textbook written by Kevin M. Lynch and Frank C. Park that provides education towards calculating a manipulator's kinematics and all its required components [9]. Topics include but are not limited to forward, differential, and inverse kinematics, trajectory generation, motion planning, and robot control. Video explanations are provided online for a more visual explanation, as well as a GitHub repository of Python and MATLAB code demonstrating these techniques [9].

Forward kinematics uses known joint angles to determine the resulting position and orientation of the end-effector. Each set of joint angles will result in exactly one solution for the end-effector's position and orientation.

Inverse kinematics uses a desired position and orientation of the end-effector and calculates the required joint angles. Unlike forward kinematics, a desired end-effector position has infinite solutions of joint angles.

Differential kinematics calculates the relationship between the joints' angular velocity and the end-effector's linear and angular velocity. This helps control the speed of the end-effector as it travels through a trajectory, such as the foot of the legs while the robot is walking.

5.3 Unified Robotics Description Files (URDFs)

Extendable Markup Language (XML) files are very common in software and are widely used to define and store data that is commonly shared among engineers [10]. Data stored in .XML files are marked with tags that describe what the data represents [10]. Additionally, there are rules in place to standardize all .XML files to improve transmission and shareability between different computers and users [10]. The .XML file does not do any computation itself but is compatible with external programming languages using the stored data with the .XML file [10].

A Unified Robotics Description File (URDF) is a variation of an .XML file, where the stored data represents the 3D information regarding a robot's joints, links, mass, motors, etc [11]. A URDF can inform an engineer about the robot's size and capabilities, 3 as well [11]. However, a URDF can also be imported into simulation, where code can be tested to evaluate the robot's behavior before it is run on its real-life counterpart [11].

5.4 Simulators

A large part of this project is the development and implementation of the robot in simulation, therefore picking the software in which the simulation will be run is extremely important. We primarily analyzed two papers, “How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion” and “A Comparison of Humanoid Robot Simulators: A Quantitative Approach”, with hopes of gaining more insight into whether we should transfer from CoppeliaSim or not. In the first article [12], Farley et al. discussed the simulators CoppelaiSim, Gazebo, MORSE, and Webots, specifically focusing on the accuracy of the motions. While focusing on accuracy, it also ranked them based on other considerations such as whether it is open source, has ROS compatibility, what programming languages it supports, the functionality of the UI, and the different model formats it supports. The authors created a sample terrain of various textures in real life with an actual robot and then attempted to duplicate it all in the various simulators, having the robot move across the terrain in both and compare sensor data. Based on their metrics and analysis, CoppeliaSim was their pick as it was comparatively accurate among different tests, is free to use, and has numerous features. The primary drawback they focused on, which is not of great concern for this project, is that CoppeliaSim is not open source. The authors also noted that it ran while utilizing significantly less computational power, allowing more complex simulations to be run on weaker machines. This was also noted by the previous year’s MQP team. As for the second article [13], Ayala et al. compared Gazebo, Webots, and V-REP (with V-REP effectively being a slightly older version of CoppeliaSim). This article compared them with metrics such as CPU usage, memory footprint, and disk access. They measured this by running a simulation 20 times and tabulating the results. They found that Webots had the lowest usage, followed by V-REP, then Gazebo. These results regarding the computational efficiency of the simulators were in agreement with those of the first article, however, we determined that this increase in efficiency is not sufficient to make it a better choice than CoppeliaSim, especially since much work for this project has already been completed in CoppeliaSim and we have not encountered any issues with our computers being able to run the simulations we create. Given that, the literature agrees with last year’s and our choice to continue our usage of CoppeliaSim.

5.5 Zero Moment Point

While the robot is moving without assistance, there needs to be an increased amount of control applied to the system. This section describes a hypothetical point known as the Zero Moment Point (ZMP), a point where all the moments around this point sum to zero. As described below, the ZMP is important to understanding the robot's stability and behavior during unassisted movements such as walking.

5.5.1 Static vs. Dynamic Balancing

When working with bipedal robots, there are two different methods to ensure that the robot is stable and balanced. Static balancing is based around the robot's center of mass and is utilized when gravity is the only force acting on the robot when the robot is not moving. In this state, the objective is to ensure that the robot's center of mass is within a space known as the support polygon [12]. The support polygon is a shape defined by the robot's points of contact with the floor.

When the robot begins to actuate any of its motors, the robot begins to experience internal forces that will affect its balance. These forces are due to the acceleration of the robot's center of mass, as well as the inertia of the robot as a whole. During movement, keeping the center of mass within the support polygon is not enough for the robot to remain balanced. Therefore, the robot must use a method to remain dynamically stable, meaning it can maintain its balance when other forces beyond gravity are applied to the robot.

For a robot to become dynamically stable, the robot must keep its zero moment point within its support polygon. The zero moment point (ZMP) is a hypothetical point on the support polygon where all the moments around the point sum to zero. With the ZMP within the support polygon, the robot is able to actuate its joints freely, allowing the robot to walk and gesture while balancing unassisted.

5.5.2 Calculating the Zero Moment Point

Unfortunately, calculating the zero moment point is not an easy task, as it requires information regarding the robot center of mass's location, linear acceleration, and angular acceleration [14]. Equations (1) and (2) are to calculate the x and y coordinates of the zero moment point.

$$x_{ZMP} = \frac{\sum_{i=1}^{n}(m_i(p_{ix}(\ddot{p}_{iz} + g_z) - p_{iz}(\ddot{p}_{ix} + g_x)) - I_{iy}\omega_{iy})}{\sum_{i=1}^{n} m_i(\ddot{p}_{iz} + g_z)} \tag{1}$$

$$y_{ZMP} = \frac{\sum_{i=1}^{n}(m_i(p_{iy}(\ddot{p}_{iz} + g_z) - p_{iz}(\ddot{p}_{iy} + g_y)) - I_{ix}\omega_{ix})}{\sum_{i=1}^{n} m_i(\ddot{p}_{iz} + g_z)} \tag{2}$$

In these equations, *m* denotes the mass of the link, *p* denotes the x, y, or z coordinate of the link's center of mass, *I* denotes the inertia tensor of the link, and *ω* denotes the angular velocity of the link's center of mass. The summation is for each link in the entire system.

There is another way to calculate the zero moment point, by measuring the robot's center of pressure (CoP) [14]. The CoP is the sum of the normal forces acting on Ava's foot concentrated on a point [14]. The CoP has the combined forces, but no moment. Therefore, the robot will be dynamically stable if the ZMP is the same as the CoP. By calculating the location of the CoP and determining if it is inside the robot's support polygon, then we can also determine if the robot is dynamically stable. Below are the equations to calculate the CoP [14].

$$F_{RN} = \sum_{i=1}^{n} F_{Ni} \tag{3}$$

$$p_{CoP} = \frac{\sum_{i=1}^{n} p_{FNi}F_{Ni}}{F_{RN}} \tag{4}$$

5.6 Conclusion

Using the knowledge gained from our literature review surrounding stability, simulators, zero moment points, and gaits, we implemented these topics throughout the academic year. Stability and zero moment points were important to achieve the simulation feats described in Chapter 10. The research regarding robot simulation software aided in the use of CoppeliaSim, described in Chapter 10. Lastly, our understanding of gaits was very useful for implementing assisted walking, as described in Chapter 10.

6. Methodology

This chapter outlines the progression of our project throughout each term, beginning with analyzing the state of the robots and the progress made by the previous year's team. The initial steps involved recreating robot movements in simulation through the utilization of CoppeliaSim and understanding the robot's circuitry.

6.1 A Term

6.1.1 Humble Beginnings

When we initially acquired both robots, Ava and Koalby, their conditions were severely compromised, exhibiting extensive damage. Koalby suffered from broken limbs and central wiring issues, while Ava, though in relatively better shape, lacked any wiring and also had a few broken limbs. You can see the condition of Koalby in Figure 6.1 In light of these challenges, we outlined several goals for the hardware aspect during A term: addressing the broken limbs, enhancing the circuitry design, and subsequently integrating the upgraded circuitry into both Koalby and Ava. We also wanted to review where the last team left off. We read both previous reports written by the previous teams, which are discussed in chapters 2 and 3.

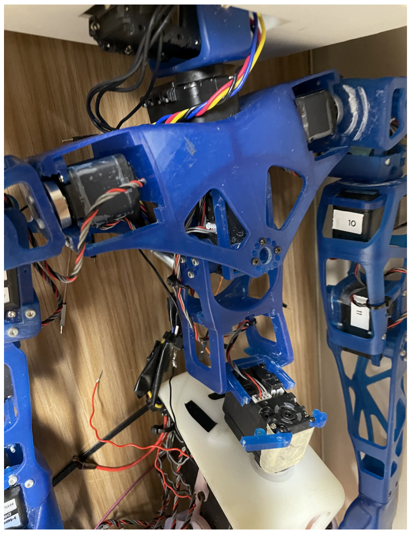

Figure 6.1 Koalby with Broken Wiring

6.1.2 Replacing the Broken Limbs

As mentioned before, many of Koalby's and Ava's limbs were broken when we received them. We wanted to get these fixed as fast as possible by learning how to use both resin and FDM 3D printers. With the help of James Van Milligen, a Graduate Student in ME, we learned how to use both printers and begin the process of replacing all the broken parts for both Koalby and Ava.

6.1.3 Circuitry Integration

Before we began iterating on the circuitry design, we had to assemble and integrate its basic components in Ava. Specifically, to integrate any circuit in Ava, we had to first accomplish the following tasks: solder, assemble, and connect the five voltage converters; solder and mount the power switch; power and connect the microcontrollers; and mount the PCB to allow for secure and organized wire splits some of which can be seen in Figure 6.2.

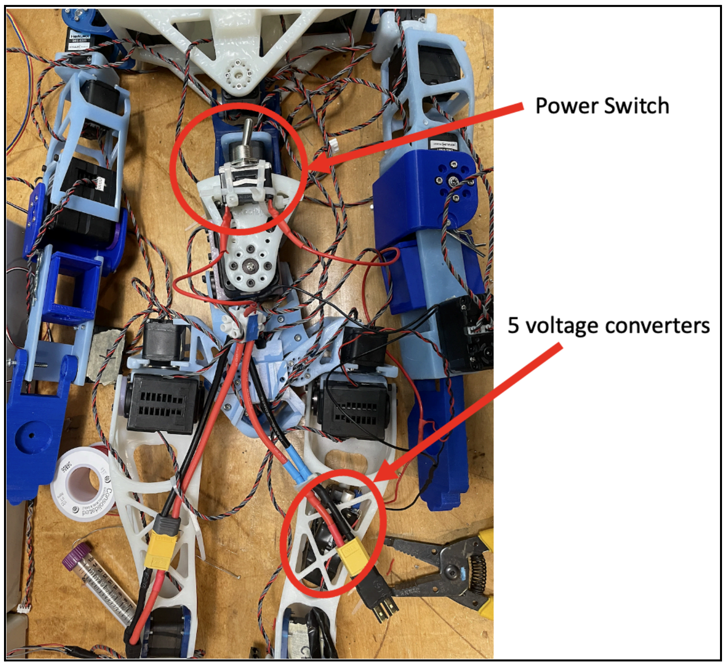

Figure 6.2 Back of Koalby with Power Switch and Voltage Converters

To accomplish these tasks, we relied on the electrical diagram from the previous year Figure 3.3. This comprehensive diagram served as a guide, illustrating the wiring configurations for each sensor and motor. Leveraging this diagram, we generated detailed wiring diagrams, Figures 8.3 and 8.4, that guided our efforts throughout the term. These diagrams were not only crucial for the successful completion of the outlined tasks but also served as a foundation for ongoing testing and improvements.

We successfully developed an organized and comprehensive wiring diagram based on the information obtained from the previous electrical diagram and subsequently implemented it in Ava. Initially, our plan involved placing a PCB board on the back of Ava. The intended configuration included having all the 0201 and 0101 motors in a chain, and similarly, the 0601 motors in a chain. However, as we proceeded with this wiring plan, a critical issue emerged. The configuration led to voltage drops across the motors, resulting in insufficient voltage for a majority of them, causing improper functionality. Faced with this challenge, we decided to shift our approach to the wiring configuration to rectify the voltage drop issue and ensure optimal motor performance. This is further explained in Chapter 8.

6.1.4 Learning and Improving Ava Within CoppeliaSim

Based on our research from our literature review and the decision matrix we created, which can be found in Table 6.1, we decided to continue with last year's usage of CoppeliaSim. We analyzed 5 different metrics, gave them weights of 1-5 and scores for each simulator from 1-5 with 5 being the best/most important. The previous team left behind a simulation of Koalby that allowed for easy testing of kinematics and joint control. However, none of the members of the 23-24 team had any experience with this software. Therefore, A term was spent learning how to control the simulation within Python. We met with members of the previous team to learn how to use CoppeliaSim, as well as completed the tutorials provided by CoppeliaSim. Afterward, we wrote PID controllers to control the angle of each joint within the simulation. We built upon the PID code to generate a time-based quintic trajectory to move entire limbs smoothly.

Table 6.1: Decision Matrix for Humanoid Robotics Simulators

Simulator	Cost (5)	Ease of Use (2)	OS Support (4)	Familiarity (3)	Software Flexibility (2)	Totals
CoppeliaSim	5	5	5	3	4	72
Gazebo	5	3	1	4	3	53
MORSE	5	1	3	2	1	47
Webots	5	4	5	1	2	60

6.2 B Term

6.2.1 Circuitry Iteration

Once the basic components of the circuit (power switch, voltage converters, etc.) were integrated, we began iterating on the design of the circuitry. The circuitry had to be able to supply sufficient voltage and current to each motor and facilitate serial communication between them and the microcontrollers, and we iterated upon the circuitry design until it adequately met these requirements.

6.2.2 Simulation - Creating and Importing a URDF of Ava into CoppeliaSim

With a new understanding of how CoppeliaSim works and interacts with the remote API, our next goal was to improve the accuracy of the simulation to behave similarly to the real robot. A full CAD model of Ava was provided by another team in SolidWorks. Using an add-on built by ROS, we could export a Unified Robotics Description File (URDF) from the assembly file. To do this, we needed to define the position and orientation of each joint's coordinate frame using reference geometry within the assembly file. We chose an arbitrary point on the back of the robot's chest to be the origin of the robot and for reference when building the URDF. The axes of the origin have the positive x-axis pointed along the left arm, the positive y-axis pointed upwards towards the head, and the positive z-axis pointed forward towards the front of the robot. Some examples of axes are shown in Figures 9.1, 9.2, and 9.3. To simplify the kinematics calculations, each coordinate frame was positioned on the center of the front face of the motor, and the orientation is the same as the origin's orientation.

With all the reference frames defined, we built the URDF with the add-on tool by defining the boundaries between each link and defining each joint as a revolute joint rotating about its respective axis. SolidWorks utilized the mass properties defined within the model to calculate each link's mass, center of mass, and inertia tensor. The add-on exported a folder that contained the URDF, but also .stl files to import into CoppeliaSim, as well.

Inside CoppeliaSim, we opened an empty scene and used the native import tool to import the URDF and .stl files. CoppeliaSim then builds the robot and defines all joint/link relationships. The last step is to define each joint's angle limits and to add the maximum velocity and torque each joint can produce.

6.2.3 Kinematics Calculations

Since we are working with Ava and not Koalby, many of the kinematics the last team calculated were inaccurate to the updated model. Therefore, we had to recalculate the kinematics for all of Ava's limbs.

We decided to move away from using Denavit-Hartenberg parameters to calculate the kinematics of the robot, a technique used by the previous team. Instead, we calculated the twists of each joint in a kinematic chain to calculate the forward, inverse, and velocity kinematics. There were multiple reasons for this change. Firstly, Denavit-Hartenberg parameters were difficult to reliably find the accurate positions and orientations of each coordinate frame, due to the high number of degrees of freedom. Additionally, if a new design was created, then the parameters would need to be recalculated and the inverse kinematics would need to be reevaluated completely from scratch. By using twists to calculate the kinematics, we were able to quickly redo calculations during design changes, as only one twist would need to be changed as opposed to reassigning every coordinate frame. Further explanation of our calculations can be found in Chapter 9.

6.2.4 Unassisted Standing in CoppeliaSim

With the new simulated model of Ava imported inside CoppeliaSim, our next goal was to implement an algorithm to allow Ava to stand without assistance in CoppeliaSim. First, we created a sensor inside Ava's chest to act as a replication of the IMU inside the real robot's chest. The sensor reports data over the remote API regarding its orientation and angular acceleration, just like the real IMU sensor we used. In each loop in the Python script, we request data from the IMU to use in the unassisted standing script.

Due to the design of the robot, when the joints inside the chest are set to zero, the center of mass of the whole system is within the robot's support polygon, a shape defined by the robot's point of contact with the ground. When the center of mass is within the support polygon, the robot is known to be statically stable, meaning the robot will remain upright and stable when the only force acting upon it is gravity. Therefore, we created a PID controller to ensure that the robot's chest remains upright and does not fall over. When the robot is standing upright, the IMU values for orientation are zero. Therefore, we created a PID controller that used the IMU values as the input and moved two joints in Ava's hip to keep the IMU values as zero. The rest of the motor's positions were set to zero.

6.2.5 Real-Time Center of Mass Detection

During B term, we created an algorithm to detect the location of the center of mass of the entire system. To calculate the center of mass of a system, we located the center of mass of each link using forward kinematics. The location of the center of mass was with respect to the robot's coordinate frame defined on its back, as described in Section 6.2.2. With the knowledge of each link's CoM's location and the mass of each link, we calculated the coordinates of the system's whole center of mass. Equations can be found in Chapter 11.

Currently, this script is not used as it is not needed in unassisted standing or assisted walking. However, the position, velocity, and acceleration of the center of mass are required to calculate the zero moment point of the robot, which is vital for unassisted walking.

6.2.6 Koalby to Finley

While collaborating with the Design team, their initial concept involved creating a third robot named Finley. This idea aimed to enhance the project by providing three operational robots

(Koalby, Ava, and Finley) to work with, thereby alleviating the issue of waiting for one robot to be repaired before progressing. However, during the process of sourcing materials and 3D printing parts, it became apparent that constructing a third robot would be expensive and extremely time-consuming. As mentioned before, our team primarily focused on working with Ava, as she was in the best condition among the two robots. As a result, the decision was made to repurpose Koalby into Finley. Koalby was then handed over to the Design team for repairs and transformation into Finley. You can learn more about their progress on the Design team's report and see Koalby's new look as Finley in Figure 6.3.

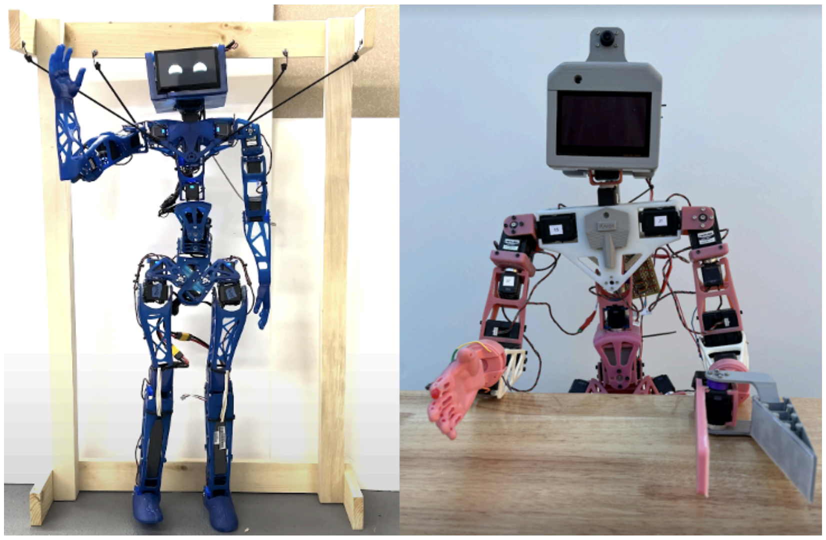

Figure 6.3 Koalby (left) turned into Finley (right).

6.3 C Term

6.3.1 Finalizing the Circuitry

Upon completion of the work on the power data distribution board, it became evident that certain motors were not receiving adequate power, or any power at all in some cases. During the process of finding out what was wrong with our electrical system, we underwent a plethora of debugging steps. The first step we took was testing the continuity between critical elements of the circuitry, including motor wires, microcontroller wires and headers, circuit board connectors, and circuit board traces. Once we inspected the circuit board, we found a few traces that may have been delaminating from the board. This issue likely stemmed from the extensive iterations the board had undergone, involving multiple rounds of soldering and desoldering, leading to trace damage. Recognizing the persistence of the problem, a decision was made to employ a new circuit board. Therefore we created a new circuit board, with the trace connections on the back of the board rather than the front to try and circumvent some of the wear and tear they may face as a result of being on the front face; this can be seen in Figure 6.4 and Figure 6.5. Additionally, through evaluating each of our designs, which can be found in the decision matrix in Table 6.2, we concluded that utilizing JST headers on the board would be the best decision. We analyzed 4 different metrics, gave them weights of 1-5 and scores for each simulator from 1-5 with 5 being the best/most important. Following the wiring of the new board, it was observed that all motors were now receiving the correct voltage, resolving the power supply issues.

Table 6.2: Decision Matrix for Circuitry Options

Solution	Ease of Debugging (4)	Functionality (5)	Durability (5)	Ease of Assembly (1)	Totals
Soldered 5-way split with daisy chains	2	3	1	4	32
Single daisy chain	4	1	5	5	51
Circuit Board With JST	5	5	5	2	72
Circuit Board Solder Connections	2	5	2	3	46

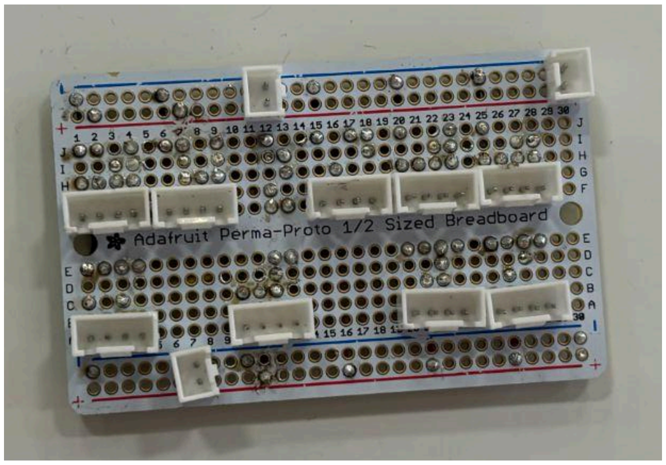

Figure 6.4: Front of Power Data Distribution Board

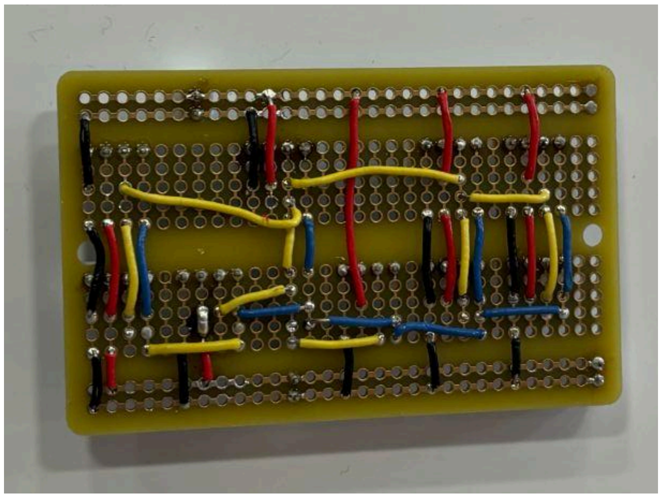

Figure 6.5: Back of Power Data Distribution Board

6.3.2 Raspberry Pi and IMU

Our main goal for this term was to install the Raspberry Pi onto the robot and enable it to send code to the Arduino. First, we set up Secure Shell Protocol (SSH) to allow for remote connection to our Raspberry Pi. Once accomplished, we connected the Raspberry Pi to our IMU, which is a BNO055 Adafruit 9-DOF Absolute Orientation IMU placed inside Ava's chest, as shown in Figure 6.6 below.

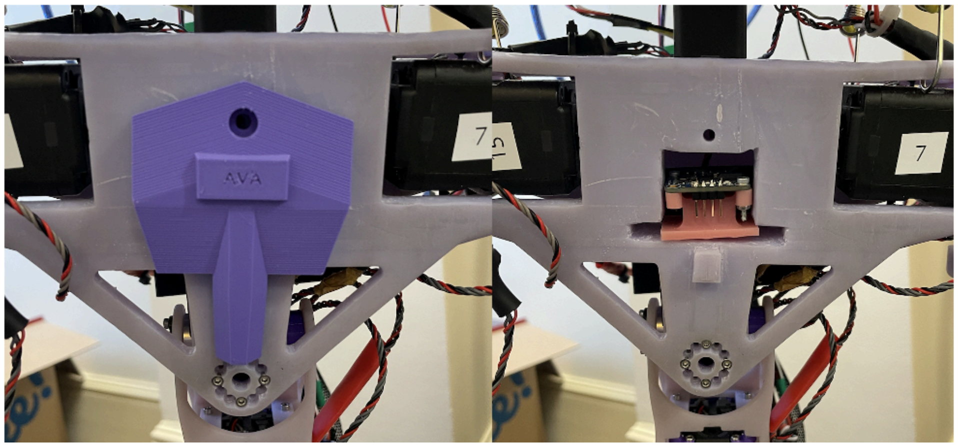
Figure 6.6 Ava’s Chest with cover (left) and without cover (right) for IMU

We successfully created wiring for the IMU and connected it to our Raspberry Pi. Subsequently, with the IMU able to transmit information to the Raspberry Pi, we established a connection between the Raspberry Pi and the Arduino. With everything connected, we can upload code to the robot. The IMU sends its orientation information to the Raspberry Pi, which then forwards commands to the Arduino. The Arduino, in turn, sends commands to the motors. The motors relay their angle positions back to the Arduino, which then transmits the data back to the Raspberry Pi; you can see the system architecture in Figure 6.7. This is further explained in Chapter 12 of the document.

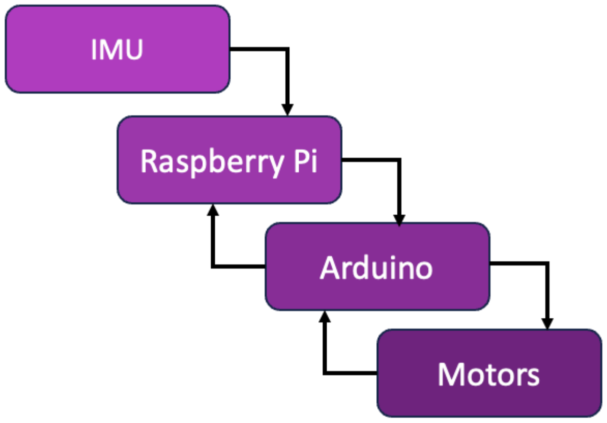

Figure 6.7 Software Architecture

6.3.3 MATLAB Scripts for Kinematic Calculations

With the kinematics of the robot calculated, we created MATLAB scripts to quickly calculate the forward and inverse kinematics of each limb. The purpose of these scripts was to calculate the via-points for each point throughout a trajectory.

6.3.4 Motion Progress in CoppeliaSim

C term was dedicated to making Ava walk in CoppeliaSim. First, a simulated cart was created by using a simple cuboid with joints attached to cylinders to act as wheels. Ava's arms were placed onto the cart to help with stability. However, due to limitations within CoppeliaSim, both arms could not be attached to the cart simultaneously. To work around this issue, we attached Ava's chest to the cart within the simulation to simulate Ava 'grabbing' the cart.

Using the MATLAB scripts we developed, we created trajectories for each leg to move through to simulate the robot's gait. These allow the robot to push itself forward.

In addition to walking, unassisted squatting was also implemented in CoppeliaSim. This was an excellent test to ensure that the active counterbalancing was working properly, as her CoM changes throughout the motion. The squat is performed by gradually moving her arms forward while also bending at the knees and ankles to act as a human-like squatting position. She then traveled from that position and back to standing in a loop to form her squatting motion.

6.4 D Term

6.4.1 Circuitry Maintenance

Because of design and iteration work on the circuitry from previous terms, the circuit did not require any further redesign in D term and largely met our overall requirements for reliability. However, several issues within the circuit presented themselves regardless throughout the early weeks of the term. These issues stemmed from wear and tear on the HerkuleX motor cables (described in Section 8.3.3), which could simply be resolved by replacing the problematic cables. Once these issues were resolved, we improved the circuit's organization through wire management, and then no other major changes to the circuit were necessary.

6.4.2 Simulation Updates

As more motion testing on the real robot began, it became evident that certain joint axes contradicted the axes set in the simulation. The reason for this was because the motor rotated with the link, meaning the joint axis would be pointed in the reverse direction than what was expected. These motors included the rotation motor within each bicep and both motors in the pelvis that control Ava's torso. Additionally, we found that some motors were put on backward during reassembly, meaning that the motor rotated in the opposite direction compared to the simulation and the CAD design. All inconsistencies were corrected in the simulation to ensure an accurate representation of the real robot.

6.4.3 Locomotion Progress

With Project Presentation Day rapidly approaching, we sprinted towards creating scripts for Ava to stand unassisted with balancing and walk while pushing her cart. After debugging motor and communication issues, we were able to start finalizing our trajectories and testing our movements. With time running out, we decided to focus on assisted walking rather than unassisted standing with balancing as we believed this goal was more achievable and would make for a more interesting demonstration. After rapid iteration of Ava's position over the cart and her leg trajectories, we were able to get Ava to walk while pushing the cart for 2 meters before stopping her manually, as shown in Figure 6.8.

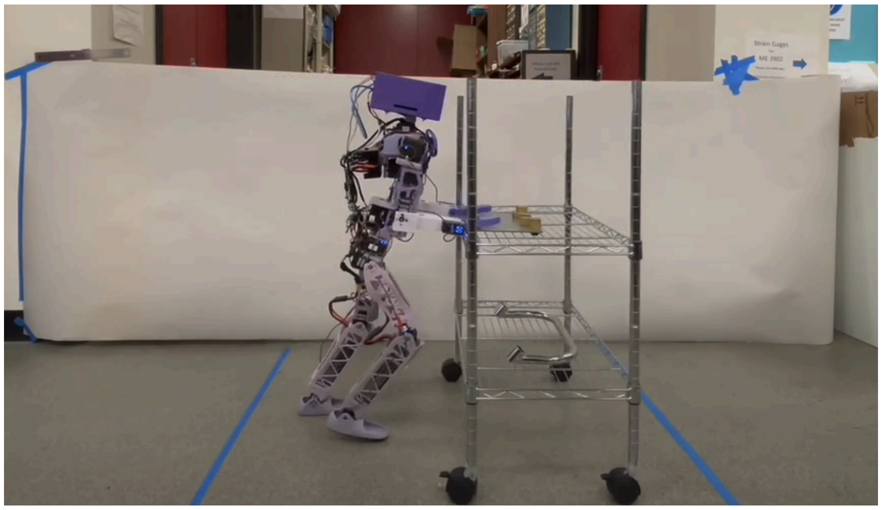

Figure 6.8: Screenshot from Video of Ava Pushing Her Cart [27]

6.5 Conclusion

Our methodology for each term reflects the progress made in the previous quarter, as we modified our plan based on our current progress or hindrance from potential setbacks. Our work in A term primarily consisted of reviewing the previous year's progress, refurbishing the robots, and planning our goals and methodology for the year. In B term, we began developing initial solutions for our electrical, simulation, and locomotion challenges based on last year's work. In C term, we analyzed issues with our initial solutions and iterated to make better choices and implement brand-new capabilities for Ava. In D term, we tested and finalized our simulation and physical locomotion code to implement unassisted standing and assisted walking reliably.

In the next chapter, we will discuss the changes made to Ava's physical design over the course of this project to allow her to stand and walk without issues.

7. Robot Evaluation

In this chapter, we will outline the various parts that were printed for replacements, design changes, part breakages, etc. When discussing the replacement of parts, we will emphasize our collaboration with the design team for assistance.

7.1 3D Printing Replacements and Design Changes

Between the last project and the commencement of this project, we had to 3D print a lot of parts for robots because limbs would break constantly. Parts would break for multiple reasons such as too much stress, falling due to instability, and storing the robot incorrectly. This led to spending countless hours replacing parts on the robot, especially during the beginning of the project. Some parts would break more often than others, which would cause major roadblocks to the project because if a part broke it would slow down our progress on working with the real robot. We decided to focus on redesigning the parts that constantly kept breaking such as the shins of the robot shown in Figure 7.1.

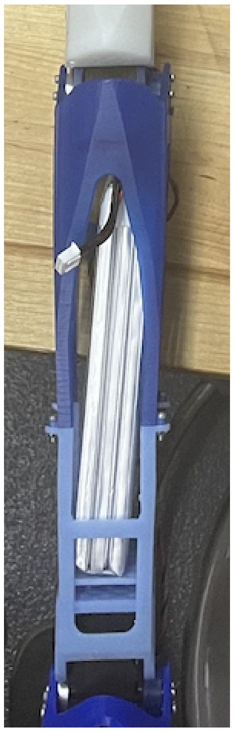

Figure 7.1 Koalby's and Ava's Old Shin design

The shin piece would constantly break because it was designed as two separate parts. In Figure 7.1, you can see that the top part and the bottom parts are colored differently. The shin was originally designed as two parts because the resin printer they had was too small to fit the design of the shin, thus the design team had to split the shin in two. However, this year we were lent a bigger resin printer, the Elegoo Jupiter 3D resin printer, which was able to fit the full shin design. We also requested the design team to design a new shin for Ava and they created this new shin part shown in Figure 7.2.

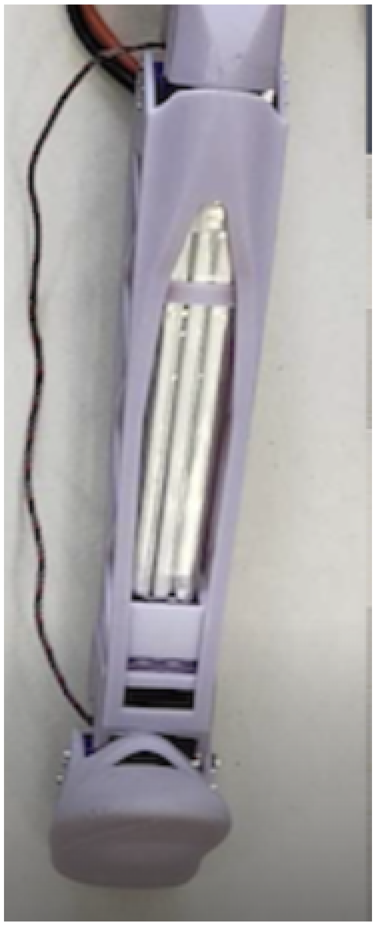

Figure 7.2 Ava’s New Shin Design

After this new redesign, the shin was no longer breaking and we were able to work on Ava more comfortably without the worry of her breaking. The shin was not the only part that was constantly breaking; parts like the: shoulder, hips, pelvis, etc. would also break repeatedly and with the help of the Design team, we were able to get those parts redesigned and fixed. In Figure

7.3, you can see the pile of broken/replaced parts that have been accumulated at the end of D term 2024.

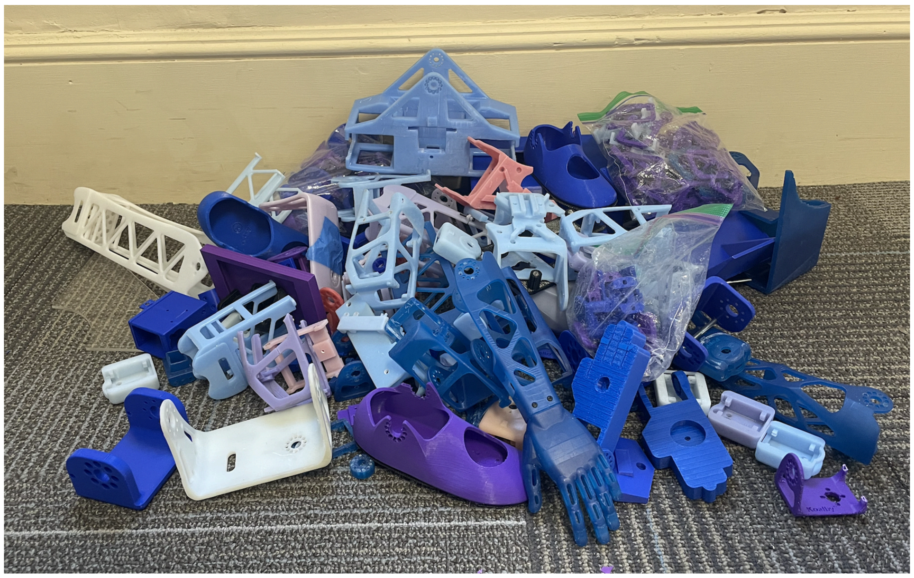

Figure 7.3 Pile of Broken/Replaced Parts

7.2 Wiring Breakages and Design Changes

Throughout the year, we have had a lot of challenges with circuitry robustness and reliability. This primarily manifests as either stress breaks of solid core wires or stress breaks where a stranded wire meets its header. In addition, we had to deal with traces delaminating however this was a more isolated problem and not recurring like our issues with wires.

Our custom cable design, as seen in Figure 8.5, is the current solution to an iterative process, making the cables stronger and more robust over each iteration. Our first cable design was naive, simply using the provided HerkuleX cabling, and jumping them to the circuit board using solid core wire. While this solution was effective in the short term, it is evident that it was insufficient long term, and unbeknownst to us, it would cause more problems than it would fix. Firstly, because of the introduction of solid core wiring into our electrical system, there was immediately a point of failure added, considering the property of work-hardening over time leading to breaks in the wire. This was the primary fault of the design. We found that using the solid core wire as a jumper between the motor chain wires and the circuit board allowed the gauge of the wiring to expand the intake of the headers, causing these cables to now be loose in their connection. This caused intermittency in our power and data delivery through these connections. Unfortunately, the only way of fixing this was to replace those cables.

This led to us brainstorming a more permanent solution, which would foster the reliability and organization of our wiring system. This was the inclusion of custom cabling, this cabling would act as an extension to the previously mentioned HerkuleX motor chain wires, and a way to facilitate a reliable connection to the power data distribution board. The introduction of JST-XHP cable headers was fundamental in removing solid core wire from our electrical solution, as they allow for crimping rather than solder or friction connections. The addition of these more robust and flexible cables as well as the implementation of disconnectable cable headers allowed us to have a more robust and less prone to failure electrical system.

The evaluation of the robot revealed frequent part breakages. Collaborating with the design team, redesigns were implemented, such as a new shin design for Ava, resolving breakage concerns. Wiring reliability issues were also addressed through iterative improvements to custom cables, transitioning to JST-XHP cable headers for enhanced robustness. These measures ensured smoother operation and laid the groundwork for a more resilient electrical system. In the next chapter, we discuss more details on circuit evaluation and how we were able to improve the wiring.

8. Circuit Evaluation and Improvements

Before any software work could be done, we first needed to ensure that the circuitry was capable of producing consistent actuation. Specifically, the circuitry needed to supply the proper voltage and current to each of the 27 servos and facilitate serial communication between each servo and the microcontrollers. Additionally, we wanted to ensure that the final circuit was functional, consistent, organized, and durable enough to withstand consistent movement without sustaining damage.

8.1 Previous Design

When considering the past design, as seen in Figure 8.1 below, we found that it did not meet three of these requirements: consistency, organization, and durability. While functional on paper, this design only worked part of the time and was clumsily wired, as shown in Figure 8.2. Upon further inspection of the previous wiring, we identified some areas of concern. There was one main five-way connection that was twisted and soldered together, causing consistency issues. This can be seen in the middle of Figure 8.2 where there is a bundle of wires zip-tied together. Additionally, the entirety of the circuit was made using solid core wire, leading to significant durability issues as the circuit bends and flexes throughout the robot's normal operation. Furthermore, we discovered areas in the wiring with exposed wires. At the top of Figure 8.2, you can see an exposed ground wire, which is not only inadequate in terms of organization but also poses a potential safety hazard if it were to come into contact with a power wire. With all these considerations in mind, our first iteration of wiring system updates began with addressing the issues of consistency, organization, and durability.

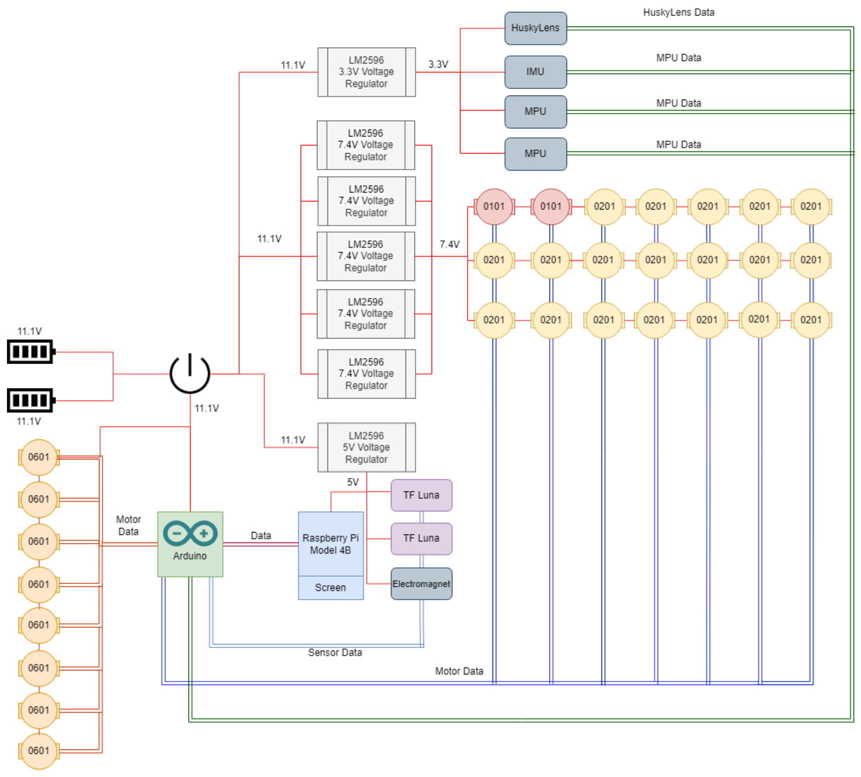

Figure 8.1: Past Wiring Diagram

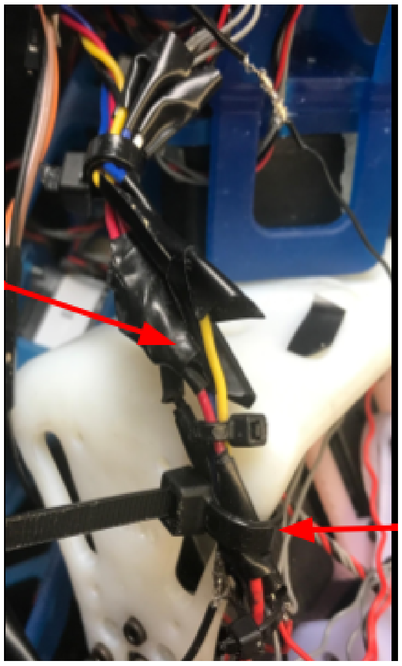

Figure 8.2: Previous Physical Wiring

8.2 Circuit Design Iteration

In our initial iteration of the circuit design, we aimed to address these issues by consolidating the previous five servo chains into just two chains: a DRS 0201 and DRS 0101 chain, and a DRS 0601 chain. The basic layout of these chains is detailed in Figure 8.3 on either side of the power data distribution board. This design successfully addressed the three shortcomings of the previous version through sheer simplicity. The straightforwardness of this design facilitated easy cable management, connecting the two chains at a central circuit board in the back of the head. This iteration also primarily consisted of HerkuleX motor wires, which are able to withstand bending much more than solid core wires. With this iteration and the implementation of a circuit board, we achieved a consistent, organized, and durable design. However, this design had a significant flaw – it was not functional. Despite being well-organized and robust due to its simplicity, along with improved consistency with the advent of the circuit board enabling secure wire connections, we were unaware of the significance of voltage drop between servos, with an average drop of 0.4V between each motor during operation. Since the servos were wired in long chains in series, the voltage drop across the servos became more pronounced as we approached the end of the chains. This resulted in insufficient power delivery to the servos at the ends of the chains which is the issue that led us to our second wiring iteration attempting to solve this issue.

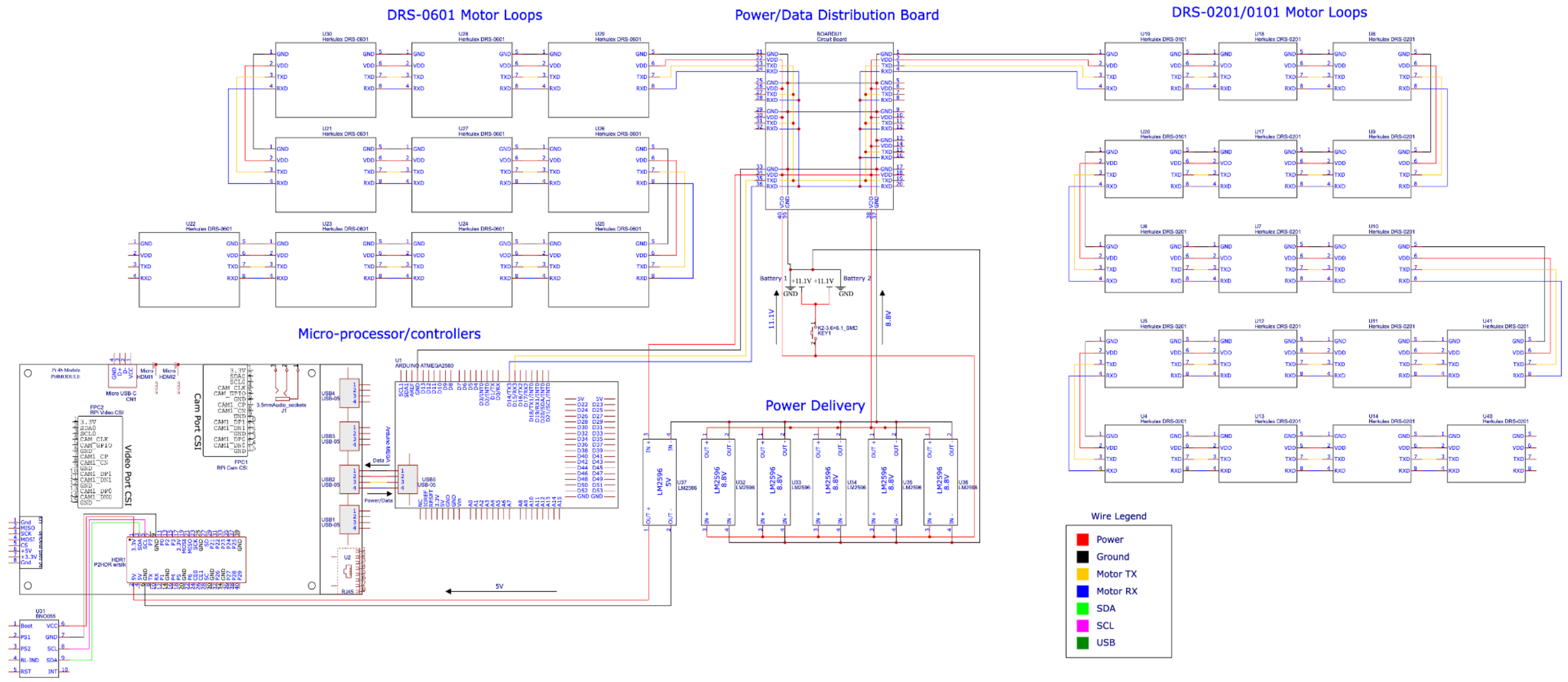

Figure 8.3: Wiring Diagram Iteration 1

When attempting to solve this problem, the first step we took was to increase the output of the voltage regulators from 7.4V to 8.8V. While this voltage remains within the threshold required for proper servo operation, it adversely affects battery life. Fortunately, we did not notice a measurable detriment to battery life while operating the robot. After testing this initial adjustment, we realized that merely increasing the output of the voltage regulators would not suffice for adequate power delivery to all servos. Consequently, we opted to rewire the servos into eight separate chains wired in parallel, ensuring that no individual chain exceeded six servos in series. This configuration marked a significant improvement over the previous design, where one servo chain accommodated nineteen servos. We initially validated this proof of concept using a breadboard. Upon confirming its efficacy, we proceeded to design a new circuit board, which is featured at the center of Figure 8.3. The board's design necessitated dual functionality: ensuring proper power distribution to the servo chains in parallel and consolidating all servo chain TX and RX UART communication lines into a single master TX and RX line. The current solution is depicted in Figure 8.4. The differences between the first iteration seen in Figure 8.3 and the current iteration as previously mentioned are the twofold. Firstly the configuration at which the motors are connected. In the previous iteration there are very long chains of motors in series connected by only one header in the power data distribution board. While the current iteration is split into multiple chains in parallel where there are no more than six motors in series. Secondly, we can see that there are two fewer DRS-0201 motors and two more DRS-0601 motors. This is a result of the knee motors being overloaded, constantly overheating. So we decided to replace them with the higher torque DRS-0601 motors to circumvent this issue.

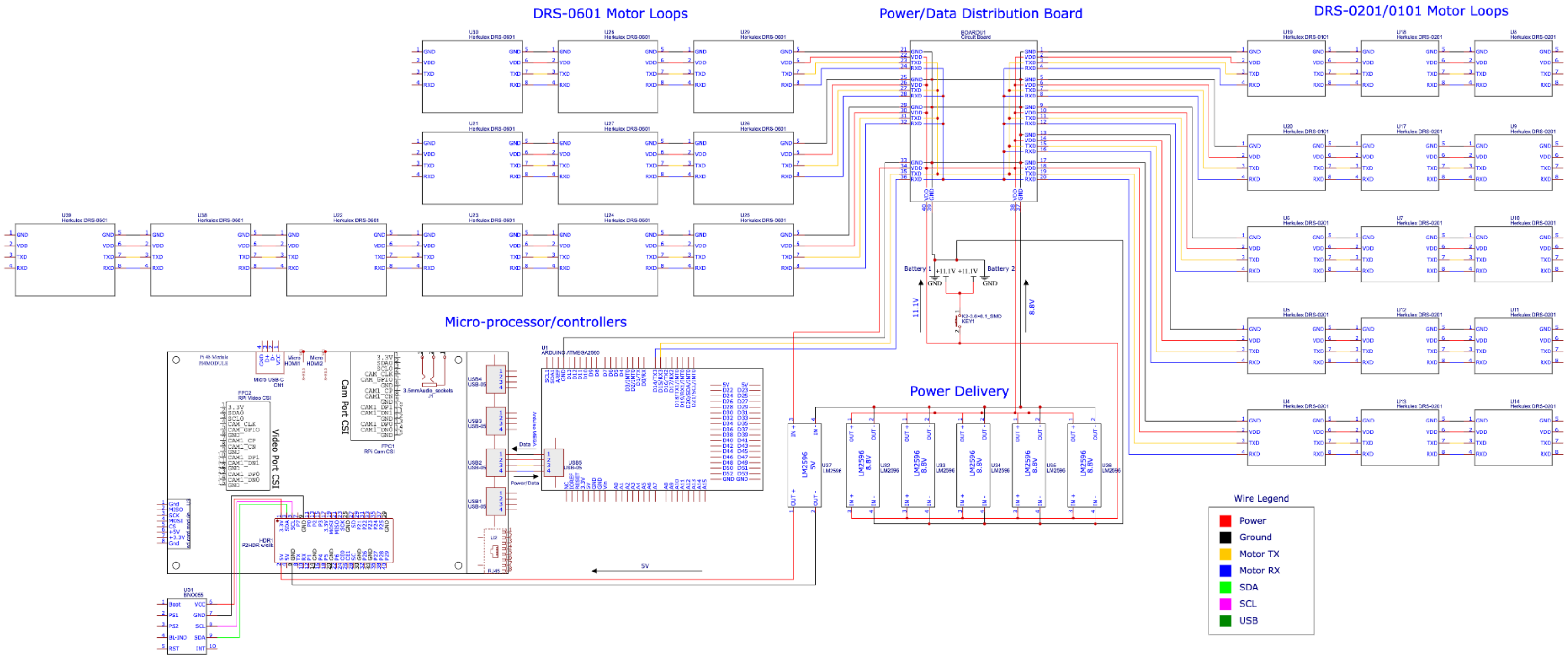

Figure 8.4 Current Wiring System Diagram

8.3 Point of Failure Reduction

In addition to changing the overall design of the circuit to work towards our goals of consistency, organization, and durability, we also carefully designed the individual components of this circuit to minimize points of failure to maximize the consistency and durability of the circuit. This section describes the design of each of these individual components and how the design works to minimize points of failure in the circuit.

8.3.1 Custom Circuit Board

The circuit board, which serves to facilitate the distribution of power and data to each chain of motors, was implemented with the primary purpose of minimizing points of failure; as seen in Figure 6.4 and Figure 6.5. The board allows the connection of a multitude of cables without requiring any multi-wire solder connection, which proved to be a significant source of inconsistency in the previous iteration.

The design of the circuit board itself also plays a crucial role in ensuring consistency. The board does not need to be flexible like the rest of the circuit (since it only needs to facilitate wire connections in a single location), so it was assembled using a solderable prototype board and fixed solid core wiring, allowing strong and consistent connections within the board.

In addition to the board itself being durable, it was imperative that the point of connection between the cables and the circuit board be extremely resilient since these connections are a very common point of failure. To ensure this connection is robust, we utilized JST-XHP cable headers to facilitate the connection. This not only ensures that the cables are unlikely to snap at the connection point but also provides the additional benefit of allowing us to unplug specific motor chains during the process of assembling and troubleshooting the circuit. This additional benefit saved a considerable amount of time during the implementation process.

8.3.2 Cable Design

In the previous year’s design, the cables leading from each motor chain to the rest of the circuit (battery, Arduino, etc.) proved to be possibly the greatest point of failure present in the circuit. As a result, we needed to redesign these cables to minimize any likelihood of a failure. Our improved cable design can be seen in Figure 8.5.

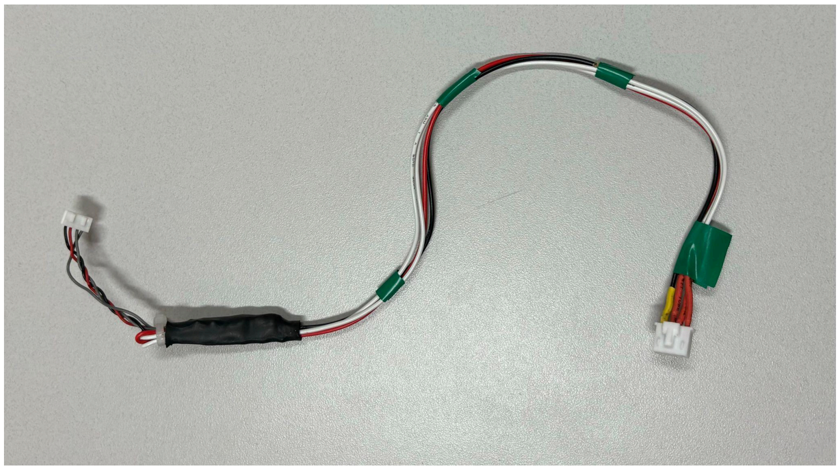

Figure 8.5: The Improved Custom Cable Design

The most significant change between this design and the previous design is the use of stranded wire instead of solid core. While solid core wire allows for easy soldering, the resulting solder joints are extremely susceptible to snapping when bent repeatedly, sometimes after being bent as few as five times. Even the wire itself, regardless of any attachment point, can break if it is repeatedly bent. This is especially problematic in the context of a 27-DOF humanoid robot, which requires bending the cables in many different locations. As a result, these cables were made using exclusively stranded wire to allow for repeated bending without any resulting damage.

Additionally, while the JST-XHP cable header helps to facilitate a sturdy connection between the cable and the circuit board, the attachment between the wire and the cable header needs to be sturdy as well. For this purpose, we employ two layers of heat shrink to reinforce the attachment point. The inner layer strengthens the connection between each wire and the header (these can be seen exposed in Figure 8.5), and the outer layer surrounds the cable itself. These both significantly reduce any folding or bending that could occur at the point of attachment, since repeated folding would result in damage long-term.

The end of the cable opposite the JST header is composed of a standard HerkuleX motor cable since it needs to be plugged directly into a HerkuleX motor at the start of a motor chain. As a result of these changes in the cable design, the cables were exceptionally reliable, resulting in no failures after they were implemented. Any failures related to the wires originated within the HerkuleX cables.

8.3.3 HerkuleX Motor Cables

The largest remaining point of failure within the current circuit is the HerkuleX motor cables themselves. These motor cables are composed of thin stranded wire connected at both ends to a proprietary HerkuleX motor cable header. Under ideal circumstances, these cables can operate with no issue, but the thin wire used, as well as the lack of reinforcement at the header connection, often leads to consistency issues.

The majority of the issues within these cables stemmed from either the wire itself snapping at the attachment point or the pin connector within the header loosening, which led to an intermittent electrical connection. Since HerkuleX motors require the proprietary header found in these cables, it is impractical to redesign them and replace them all with custom cables. As such, these inconsistencies must be minimized, rather than fully fixed.

The consistency issues present within the HerkuleX motor cables occur as a result of tension being applied to the cable, either during the robot's operation or during manual handling of the robot outside of its operation. As such, the cables were rewired to allow measurable slack within the cable at all points during its operation, preventing any tension within the cable, and minimizing any damage to it that could lead to. While these changes did not eliminate these failures, they did significantly reduce the frequency at which these failures occur, and solving them is typically as simple as replacing a cable.

8.4 Current Design

Our final circuit configuration relies on two 11.1V batteries, which supply power to the circuit board. This board then distributes power to the three DRS 0601 chains directly. Additional servo chains and certain other components draw power from five 8.8V voltage regulators. To accommodate the substantial servo load, these regulators are wired in parallel supplying sufficient current to the system. We intentionally maintained the regulators at 8.8V rather than reverting to the initial 7.4V, prioritizing consistent power delivery over extended battery life, however, after using the robot for multiple terms with this solution we did not notice any measurable difference in battery life. The Raspberry PI 4D receives power from an array of dedicated voltage regulators, set to 5V; the PI also powers the IMU through its dedicated 3.3V output pin and powers the Arduino Mega through their USB ports. Our design takes into account the communication lines of the respective components. As previously mentioned, the servo communication lines converge at the circuit board, with the servo TX connected to RX1 of the Arduino and the servo RX connected to TX1 of the Arduino. The IMU utilizes I2C communication, populating the SDA and SCL pins of the Raspberry PI. Additionally, we employed UART communication for data exchange between the Raspberry PI and the Arduino using the built-in USB ports on either device. The integration of communication lines, along with a well-tested approach to power distribution, establishes the core of the electrical system in our humanoid robot. This solution can be seen as previously presented in Figure 8.4

8.5 Conclusion

Overall, these improvements to the circuitry have either met or exceeded each of our goals and greatly improved the functionality and consistency of the robot's operation. These redesigns, as well as the integration of the custom circuit board and cables, have significantly improved the circuit's durability and flexibility, and thus its consistency as a result. These elements, in addition to improved wire management, have also significantly helped the circuit's organization. As a result, we did not experience any major circuit failures throughout the last month of the project, and Ava is now capable of consistent and reliable actuation necessary for humanoid locomotion.

9. Kinematics

Due to the high number of degrees of freedom, the forward kinematics were calculated using the product of exponentials method within the Python code. Kinematic calculations are done in each limb (each arm and each leg) to plan their respective movements during locomotion. The robot's center of mass can also be found by using forward kinematics on all joints in the system.

9.1 Defining the Body Frame

In order to calculate any kinematics within the robot, a coordinate frame must be assigned to reference the dimensions and orientations of both the end-effector and each joint in the manipulator. For kinematics regarding the arms and legs, we assigned a body frame at the end effector of each limb. For the arms, the body frame was placed in the hand's palm, concentric with the electromagnet as depicted in Figure 9.1. For the legs, the body frame was placed in the center of the bottom of Ava's foot as depicted in Figure 9.2. The orientations of the body frame matched the orientation of the model in SolidWorks, where the positive x-axis was pointed towards Ava's left hand, the positive y-axis pointed upwards towards Ava's head, and the positive z-axis was pointed towards Ava's front.

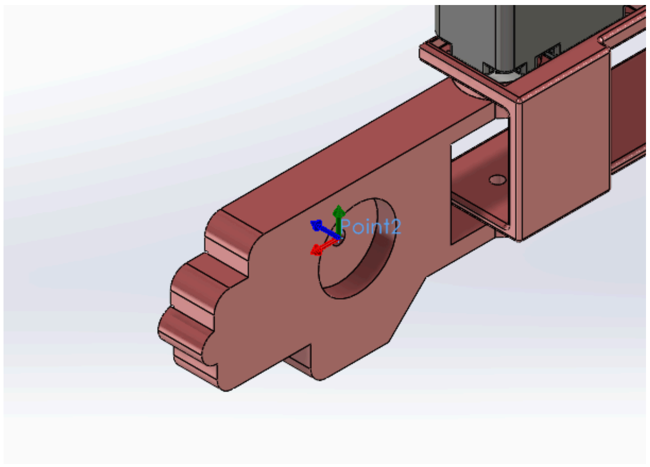

Figure 9.1: Body frame on Ava's left arm. The right arm's body frame is in the same location on the right arm, with the same orientation

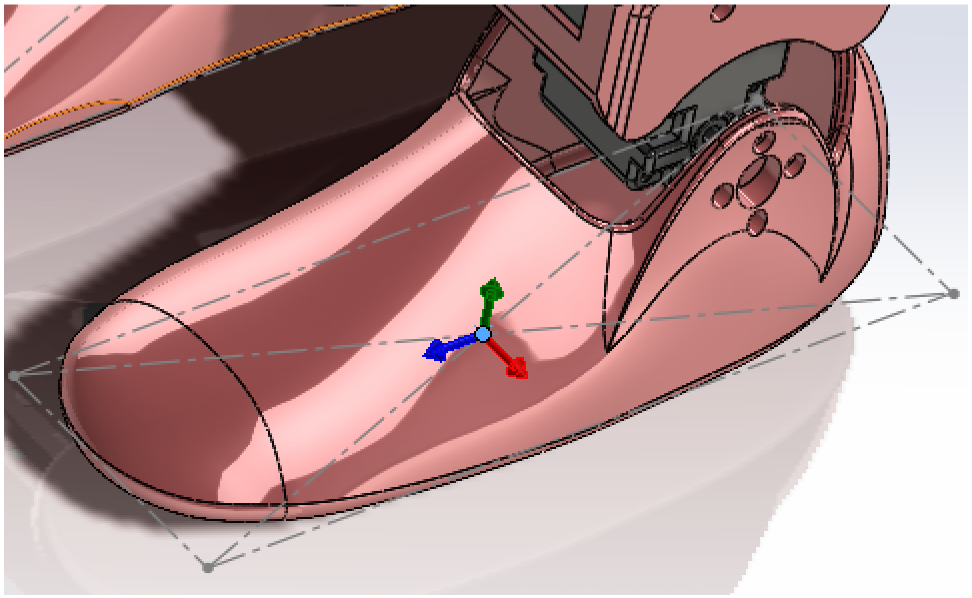

Figure 9.2: Body frame on Ava's left foot. The right leg's body frame is in the same location on the right foot and has the same orientation

By using a body frame, each motor's twist will be with respect to the body coordinate frame, as well. This allowed for easier calculations for inverse kinematics. Additionally, any calculated trajectories would also be in the body frame, allowing for the end-effector to move straight in the robot's task space.

To calculate the center of mass of the robot, we used a body frame placed in the back of Ava's chest, with the same orientation as described for previous body frames, shown in Figure 9.3.

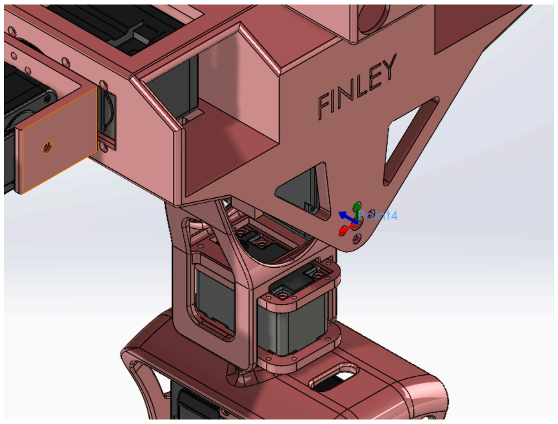

Figure 9.3: Body frame location and orientation on Ava's back, used for calculating the robot's center of mass.

9.2 Forward Kinematics

To calculate the kinematics of a limb of the robot, we must define a body frame to reference. Depending on the motion we want to perform and the limb we want to move, we have defined multiple body frames. Each arm and each leg has its own body frame to calculate kinematics with respect to. For each arm, the body frame is in the arm's respective shoulder. For each leg, the body frame is defined in the pelvis.

To calculate forward kinematics, we utilized the product of exponentials method. For each limb, we calculated the home configuration matrix M, a 4x4 homogeneous transformation matrix from the body frame to the end-effector of the limb. Then, we calculate the twists of each joint in the manipulator, representing the linear and angular velocities of the rigid body. All twists are represented in their respective body frame.

$$S = \begin{bmatrix} \omega_{3x1} \\ v_{3x1} \end{bmatrix} \tag{5}$$

With the home configuration and all the twists created, we then calculated the exponential coordinates of rotation and the exponential coordinates of rigid-body motion from each twist.

$$R_\theta = e^{[\omega]\theta} = I + \sin(\theta)[\omega] + (1 - \cos(\theta))[\omega]^2 \tag{6}$$

$$e^{[S]\theta} = \begin{bmatrix} e^{[\omega]\theta} & (I\theta + (1 - \cos(\theta))[\omega] + (\theta + \sin(\theta))[\omega]^2)v \\ 0 & 1 \end{bmatrix} \tag{7}$$

$$T_1^n = T(\theta) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} ... e^{[S_n]\theta_n} M \tag{8}$$

By inputting each joint angle to these coordinates and multiplying all the coordinates and the home configuration matrix together, we can calculate the forward kinematics of the limb.

Kinematics is vital to creating the walking motion seen in the assisted walking demonstration video. The simulation uses forward, inverse, and differential kinematics to control the limbs of the robot and create an understanding of the relationship between the robot's joint space and task space.

To assist with the calculations, the Python script uses the open-source repository provided by Modern Robotics [9]. The repository provides generic code and functions to run and test kinematic calculations that can be modified to work on different manipulators. We utilized this repository to assist with the calculations of the forward, inverse, and differential kinematics of each limb and the entire system.

To calculate the kinematics of the robot's limbs, we needed to find the home configuration and twists with respect to the body frame assigned to each limb. Since we already had the SolidWorks model of the full robot, we used SolidWorks to find the required dimensions for each joint to calculate all the twists. Appendix A contains all the twists and the home configurations for the legs of the robot.

With the twists and home configurations found and documented, it is implemented into the code under a configuration file that is referenced whenever we want to perform any kinematics calculations. Inside the Python script, we create a robot class that stores joint information in motor objects. These motor objects contain their respective twist, as well as the home configuration from the body frame to the center of mass for the link being controlled by that joint. The center of mass is used in future calculations to find the location of the entire system's center of mass using forward kinematics.

All used twists and home configurations can be found in Appendix A.

9.3 Inverse Kinematics

With the home configuration and joint twists calculated, we also calculated inverse kinematics for each limb. We used the Newton-Raphson method provided by Modern Robotics [9] to calculate inverse kinematics. The Newton-Raphson method is an iterative method that takes an initial guess of joint angles to calculate the joint angles required to reach a desired position and orientation of the end-effector. The Newton-Raphson method works by providing the manipulator's home configuration, joint twists, and a guess of all the joint angles to reach the desired end-effector position and orientation.

The process begins with performing forward kinematics with the guess angles provided. Then, the method compares the computed end-effector location with the desired location. If the error between the calculated frame and the desired frame is small enough, then the process is complete. Otherwise, we calculate the pseudo-inverse of the manipulator's Jacobian matrix, multiply it by the twist that represents the error between the two frames, and add it to the initial guess. By continuing the process until the error between the two frames is small enough, we get the required joint angles to reach the desired position and orientation of the end-effector.

9.4 Differential Kinematics

Differential kinematics is vital to understanding the relationship between the angular velocities of each joint in a manipulator and the twist of the end-effector. In order to calculate the forward and inverse differential kinematics of a manipulator, we must first calculate the Jacobian matrix of the manipulator. The Jacobian matrix establishes the relationship between the manipulator's joint space and task space. The Jacobian is used in many more applications beyond differential kinematics, such as inverse kinematics, statics, and dynamics. The Jacobian matrix can be calculated using the formula:

$$J(\theta) = \begin{bmatrix} J_1 & J_2(\theta) & \dots & J_n(\theta) \end{bmatrix} \in \mathbb{R}^{6\times n} \tag{9}$$

With $J_1 = S_1$

and $J_i(\theta) = [Ad_{e^{[S_1]\theta_1} \dots e^{[S_{i-1}]\theta_{i-1}} e^{[S_i]\theta_i}}]S_i$

With the Jacobian matrix, we can calculate the forward differential kinematics using equation 6, and inverse differential kinematics using the following equations:

$$V = J(\theta)\dot{\theta} \tag{10}$$

$$\dot{\theta} = J(\theta)^{-1}V \tag{11}$$

9.5 Trajectory Planning

Every motion that we plan using trajectories is a quintic trajectory. This is extremely important, especially for larger motions due to how joint motion can create moments on the robot. A quintic trajectory is one that allows for the fine control of position, velocity, and acceleration along all waypoints. When analyzing forces on systems, acceleration is what drives those forces, as the force generated is equal to the mass multiplied by the acceleration. When moving the limbs of the robot, it is important to keep this in mind, as a high acceleration in the limbs is more than enough to throw it off balance.

We calculate the quintic trajectories by first inputting the target times, positions, velocities, and accelerations of each part to be moved, whether joints in joint-space trajectories or the foot or hand in task-space trajectories. All that is left is to give the time that has elapsed since the trajectory started and the position targets are outputted.

9.6 Conclusion

By using these kinematic calculations, we were able to establish the relationship between the robot's joint space and task space. Therefore, we were able to quickly understand how the robot is able to move each limb. With this understanding, we were able to create trajectories to control each limb's motion during Ava's assisted walk. Using kinematics and trajectory generation, any basic movement is possible. These calculations act as a foundation to dynamics and increased control.

10. Simulation

Last year's team designed a simulated environment for the previous robot, Koalby. They decided to use the simulation software CoppeliaSim due to its compatibility with many computer operating systems. Within the simulation, the previous team built a simplified model of Koalby and applied a gyro sensor and accelerometer to simulate the IMU data. Although it created a decent foundation and learning platform, the simulation lacked accuracy compared to the real robot. The simulation did not have the correct dimensions of the updated design, lacked simulation of torque applied by each joint, some joints rotated around the incorrect axis, some joints were missing, and the mass of each link was not set within the simulation.

10.1 Building an Accurate Simulation Model

It is vital to ensure that the simulated version of the humanoid robot is as similar to its real-life counterpart as possible. Therefore, there must be a high expectation of accuracy for the parameters of each joint and link in the entire system. CoppeliaSim allows users to import a URDF containing important data such as mass, center of mass location, moments of inertia, etc. Additionally, we are able to import .STL files to ensure the simulated model has the same physical shape as the real robot [15]. Our SolidWorks files can be found within the project's GitHub repository linked in Appendix B. Using an add-on created by ROS [16], we are able to define each joint and link within SolidWorks and export a URDF and meshes to import directly into CoppeliaSim. Within the SolidWorks model, we have to define each coordinate frame for each joint, as well as the axis that the joint will rotate about. After defining all joints and links with their respective coordinate frames, SolidWorks exports a zipped folder containing the generated URDF and all the meshes for CoppeliaSim to use. After importing the file, the full model of the robot is created in the simulation, as shown in Figure 10.1.

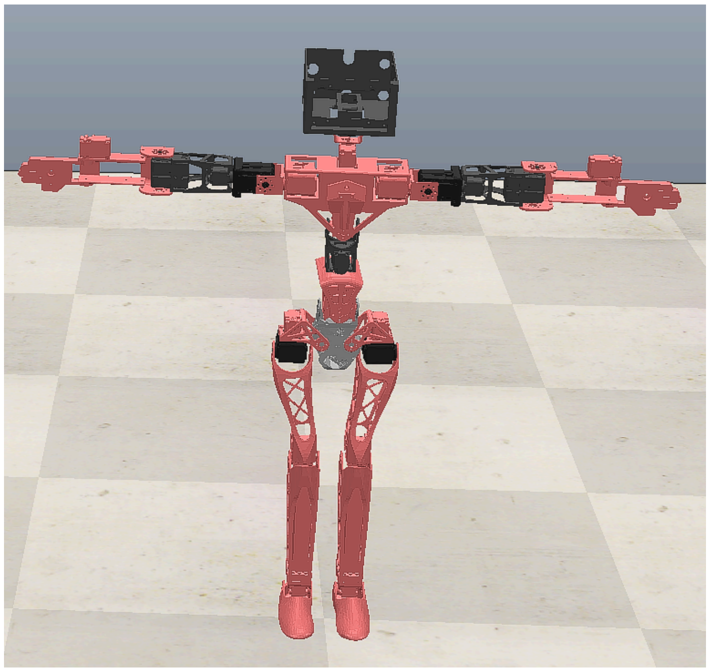

Figure 10.1: New simulation model of Ava within CoppeliaSim.

The team also improved the Python code that interacts with the simulation by cleaning up commonly used functions to read and write to the simulation's motors. Previously, the team utilized an internal PID controller within CoppeliaSim. This was built upon by creating custom PID controllers within the Python code that controls each motor. Each joint has its own PID controller object within Python that is connected to a specific motor in simulation, controlling its velocity.

10.2 Implementing Sensors

The simulated robot needs to match the number and type of sensors that are on the real robot. The servo encoders are simply to simulate, as there are API functions within CoppeliaSim that can return a motor's current joint position to Python. We were able to use this data for PID control to control each joint's position. The IMU is a combination of a gyroscope and an accelerometer. Within CoppeliaSim, there are open-source sensors that simulate an accelerometer and a gyroscope. These sensors were placed inside Ava's chest, in the same place as where the IMU is located on the real robot. Python functions were written to allow for communication between these sensors and the Python script.

10.3 Finding Ava's Center of Mass

In order for a humanoid robot to balance unassisted, its center of mass must be above the polygon defined by the surface in contact with the ground, which is the bottom of Ava's feet. It is important for Ava to know where her center of mass is at all times. We already had the information regarding each link's mass and their center of mass location with respect to the body frame when all joints are at 0 degrees. To find the center of mass of each link, the code uses forward kinematics with the link's center of mass as the "end-effector" to find the center of mass's location with respect to the body frame. This process repeats throughout all the links. In the end, all the links' center of masses will be located with respect to the body frame. Using the equations shown in Figure 12.1, we are able to estimate where Ava's center of mass is currently located.

$$x_{CoM} = \frac{\sum_{i=1}^{N} m_i x_i}{m_{total}} \tag{12}$$

$$y_{CoM} = \frac{\sum_{i=1}^{N} m_i y_i}{m_{total}} \tag{13}$$

$$z_{CoM} = \frac{\sum_{i=1}^{N} m_i z_i}{m_{total}} \tag{14}$$

10.4 Unassisted Standing Using the IMU

When all joints are set to 0 degrees, Ava's center of mass is above her feet, allowing her to balance. Using the IMU placed inside Ava's chest, we are able to know the orientation of Ava's upper body. We implemented a PID controller to keep the pitch, roll, and yaw values at 0 radians. This controller allows Ava to recover and keep her balance from any disturbances. Combined with the PID controller controlling the motor positions to keep them at 0 degrees, Ava has been able to stand unassisted for an indefinite time in CoppeliaSim.

10.5 Active Balancing

All of our complex motions require the usage of active balancing to counter the forces caused by the other joints moving. This active balancing is achieved by constantly reading angular position data from the IMU in the chest and constantly trying to keep it oriented such that the chest is perfectly straight up. 2 PID controllers were constructed, one to control the pitch of the upper body and one to control the yaw of the upper body. The pitch and yaw are both actuated using one motor each in the hips. These motors were chosen as they are at the base of the upper body and have higher torque than most of the other motors so it is easier for them to move the significant mass that makes up the upper body. Ideally, this control algorithm would be swapped out for, or modified to be, a controller that directly analyzes the center of mass and uses a similar set of PID controllers to ensure the center of mass is over the support polygon keeping the back straight, while for our use cases is sufficient, for more advanced motions including unassisted walking, is unlikely to work. Eventually, that too, would be replaced by modulating the zero moment point in the same way, and both of those are included in future work.

10.6 User Study

In addition to developing the simulation itself, another important component we focused on was ensuring that it was easy to download, install, and use. To accomplish this, we created a basic user study. We gave five people our README found on our GitHub repository and instructed them to follow those instructions to see if they could replicate the demonstration motions on their systems. All people involved in the study were not familiar with the CoppeliaSim, and we sampled people who primarily use either MacOS, Windows, or Linux. Besides sharing some basic information such as how to clone a GitHub repository and install CoppeliaSim, no additional supervision was given.

10.7 Conclusion

As mentioned in Chapter 10.1, it is vital to create an accurate simulation of Ava to allow for rapid algorithm testing and evaluation. With a simulation, we are able to test algorithms without risking damage to the real robot via unexpected movements. We continued to use the same simulation software as last year, CoppeliaSim, but increased the simulated model's accuracy regarding its 3D geometry and mass properties. We implemented a simulated version of Ava's IMU sensor to sense Ava's orientation and angular acceleration. We wrote various Python scripts to achieve unassisted standing and active balancing. Lastly, we made the simulation code and GitHub repository open-source to allow other people outside our MQP team to download the scripts, run them on their own machines, and achieve identical results as shown in this report.

11. Locomotion Software

Last year's software for the physical version of Koalby re-implemented the basic primitive movements (waving, dancing, etc.) from the previous 2021-2022 Locomotion team, created a Flask user interface to record and playback primitive movements, and experimented with both the HuskyLens AI camera and TF-Luna LiDAR sensor for perception as a lab assistant. They made minor progress in all of these areas but did not advance the locomotion of Koalby beyond facilitating the creation of hardcoded joint motions. Much of last year's code also did not carry over from Koalby to Ava due to Ava having different types of motors in different locations on the physical robot, and we found ourselves almost starting from scratch. To create basic locomotion functionality that would allow future MQP teams to develop more specialized elderly assistive care applications, we aimed to focus solely on developing code to allow Ava to both stand without external support and walk while pushing a cart.

Through the duration of this MQP, we have designed, implemented, and tested code to perform complex motions such as unassisted standing and assisted walking with a cart reliably. We have developed robust code bases in Arduino and Python as well as several validation scripts to ensure our developed functionality is sound. Improving on last year's architecture, Ava's Raspberry Pi 4B handles trajectory generation and motion planning and sends HerkuleX motor commands to her Arduino MEGA 2560 via serial communication. Her Arduino communicates directly with the HerkuleX motors through a separate serial connection, forwarding setpoints from the Raspberry Pi to the motors and relaying actual motor angles back to the Raspberry Pi. Through extensive testing, Ava can reliably stand without any external support for upwards of 3 minutes and can walk while pushing a cart for at least 2 meters uninterrupted. All code can be found in the GitHub repositories linked in Appendix B.

11.1 Arduino Codebase

We have written firmware and various testing scripts for Ava's Arduino MEGA 2560 to interface with her Raspberry Pi and HerkuleX motors. In the process of developing Ava's firmware, we also fixed a significant issue with the provided HerkuleX library and re-factored all Arduino code to use the PlatformIO framework [24]. The firmware is designed to run alongside motion planning algorithms from the Raspberry Pi, and it functions as a middleman between the Raspberry Pi and the motors to forward motor commands and relay position data. We also created several testing/debugging scripts for all aspects of the firmware, such as checking if all motors are connected or sending all motors to their home position.

11.1.1 HerkuleX Library Fix

While developing code to allow Ava's Arduino to control her HerkuleX motors, we noticed a discrepancy in the provided HerkuleX Arduino library. This library implements UART serial communication between the Arduino and HerkuleX motors to create packets for desired commands like moving to an angle and also request and decipher received packets containing information such as the motor's actual angle. For both setting and reading positions, the library needs to convert a floating point angle into an integer value (or vice versa) due to the required packet structure. The HerkuleX register in the built-in motor controller in each motor stores the motor's current position as an integer value to save memory, so this analog-digital conversion is needed to translate a desired motor angle into the equivalent integer value.

The problem arises due to a difference in the range of values of that position register in the different types of motors we use on Ava. We use HerkuleX DRS-0601 motors for high-torque joints like those in the shoulders and elbows and HerkuleX DRS-0201 or DRS-0101 motors for places where high torque is unnecessary due to their cheaper price point. As referenced in their respective HerkuleX user manuals, the 0601 motors have a position register range of 0-2047 (11-bit unsigned integer) [26] while the 0201 motors have a range of 0-1023 (10-bit unsigned integer) [25]. While it is nice to have the 0601 motors use a higher resolution register for more a precise angle value, the HerkuleX Arduino library does not account for the different mapping between the continuous range of angles (-160.0 to 160.0 degrees) and the discrete range of position register values depending on the model of the motor being commanded. The provided HerkuleX library assumed that the motor to communicate uses the 10-bit position register of the 0201/0101s, resulting in angle values computed using the analog-digital conversion formula being exactly half of what they should be for 0601 motors while being correct for 0201/0101 motors.

To solve this problem, we decided to modify each provided library method to include an extra boolean parameter for whether the motor being communicated with was an 0101/0201 model or an 0601 model. In each method dealing with communicating motor angles, we changed the maximum value of the mapped range in the analog-digital conversion formula to be either 1023 or 2047 based on the value of our added parameter. This fix had the short-term drawback of requiring changes to many library calls in our existing firmware and debugging scripts, but it both provided a way to correctly send motor commands to each of Ava's motors and ensured that this problem would not arise in the future.

We had considered implementing ways to automatically determine the model of the motor based on certain register values/ranges. However, this method would not only take longer to implement and properly test but also add more latency to library calls when waiting on an additional serial command/response.

11.1.2 Platformio Transition

Before developing software for Ava's Arduino, we decided to transition to PlatformIO due to several benefits in the development and testing process. PlatformIO is an embedded software development tool that heavily facilitates the development, testing, and deployment processes for software on microcontrollers [24]. The PlatformIO extension for Visual Studio Code (VS Code) supports thousands of boards and tens of thousands of libraries to facilitate the creation of large codebases for embedded systems. This framework allowed us to easily create C++ projects for our Arduino MEGA 2560 which included the necessary HerkuleX libraries and could be deployed quickly for rapid testing.

Previous Arduino scripts were written in Arduino (.ino), which builds off of normal C++ to add built-in library functions specific to Arduino development like GPIO functionality. The transition from Arduino scripts to PlatformIO scripts is seamless; we create a new PlatformIO project for each Arduino script we wish to convert, copy the contents of the Arduino script into the newly created main.cpp, and add the locations of any required libraries to the project's config file (platformio.ini). Once the files are copied, we can easily build, deploy, or test our script using the PlatformIO extension for VS Code. The Arduino IDE makes it difficult to develop code across multiple files and only allows the inclusion of external libraries as local ZIP files, while PlatformIO makes it easy to write classes and functions in separate files. PlatformIO also allows the inclusion of local libraries as header/source files and even remote libraries from a link to online git repositories. These features were especially useful for developing and testing common libraries which all of our other scripts referenced.

The appeal of switching to PlatformIO was obvious due to the numerous benefits the framework provides, but this decision was not taken lightly. To determine if such a major refactoring would be beneficial in both the short-term and the long-term, we also considered any potential drawbacks. PlatformIO requires the use of VS Code, meaning students without familiarity with either PlatformIO or VS Code may struggle to set up their development environment and build off of our codebase. Robotics Engineering students at WPI use VS Code and PlatformIO in the 2000-series courses to program an ESP32 microcontroller to develop several robots, meaning future teams working on this project are likely to have members familiar with VS Code and PlatformIO. Both VS Code and PlatformIO are intuitive and have extensive documentation/tutorials, allowing new users to get quickly acquainted with the tools. Additionally, we have provided specific videos and documents to help users understand the structure of our Arduino code and how to use the necessary development tools to continue our work in future years. Given these resources, we expect future MQP teams will be able to benefit from our decision to write Arduino code in PlatformIO with little to no learning curve.

11.1.3 Firmware

We extended last year's firmware program to allow Ava's Arduino to be used in combination with her Raspberry Pi for complex motions like unassisted standing and walking. The Arduino firmware busy-waits for a serial command from the Raspberry Pi, executes the desired command, and relays any desired data back to the Raspberry Pi. Each command is a packet consisting of the ID of the command to execute and any arguments the command needs, such as a motor ID or desired angle.

We removed unnecessary commands from last year's firmware, modified the remaining ones for use with Ava's different motors, and implemented functionality for motor error status handling. Below in Table 11.1 is a brief description of each command the firmware can receive, their required arguments, and the data sent as a response back to the Raspberry Pi.

Table 11.1: Firmware Methods, Parameters, and Relayed Return Data

ID	Command	Arguments	Return
1	Initialize Motors	None	None
5	Get Motor Position	int motorID	float angle
10	Set Motor Position	int motorID, float position, int tTime	None
20	Set Motor Torque	int motorID, int setTorqueOn	None
30	Read Battery Level	None	float voltage
40	Set Motor Speed	int motorID, int goalSpeed	None
50	Check Motors Status	None	List[String desc, int stat, float angle]
100	Shutdown Motors	None	None

To make the transition between simulation and physical movements easier, we define a common home position (shown in Figure 11.1) where all motors have a position of 0. Any commanded setpoint or position readings for a motor will be relative to its angle at the home position. In the firmware, we keep track of any offset between the desired angle, 0, and the actual motor angle for each motor at its home position. It is much easier to determine or modify these motor offsets in the firmware rather than to manually disassemble and rotate each motor to be at angle 0 in its home position. However, this system means we commonly must re-calculate Ava's home offsets when replacing broken parts or motors due to motors being rotated during disassembly or re-assembly. Once Ava's motor home positions are measured and stored in a common library, her firmware can reference these values as needed.

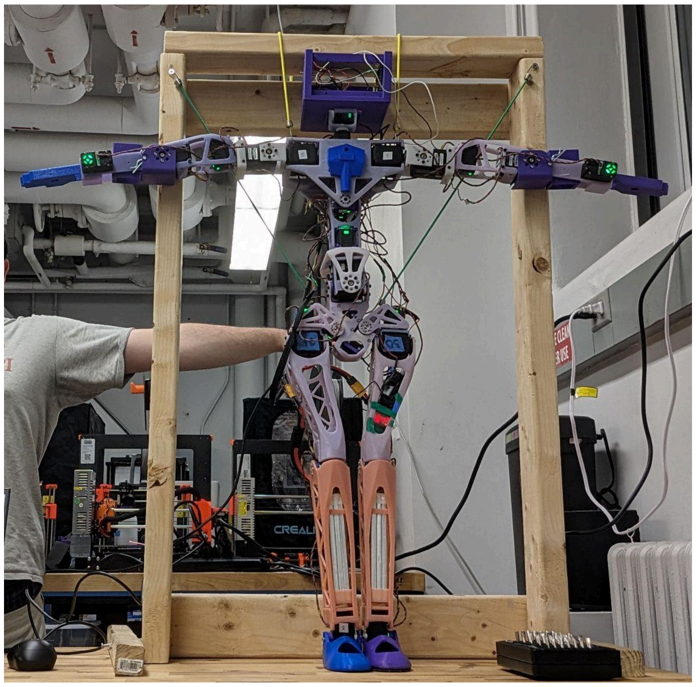
Figure 11.1: Ava's Home Position

The set of implemented firmware commands listed in Table 11.1 allows Ava's Raspberry Pi to communicate effectively with the HerkuleX motors. At the start of programs, the initialize motors (ID 1) command can be used to reboot all motors and send them to their home positions. The Raspberry Pi can read specific motor angles (ID 5), set motor positions (ID 10), and set motor velocities (ID 40) to implement closed-loop control. Our Raspberry Pi scripts can detect and handle motor errors such as a torque overload or communication error using the firmware's check motor statuses (ID 50) command, which iterates through each motor and relays the name, error status, and angle of each motor with a non-zero error status. This is useful for rebooting specific motors with errors or ending a script early due to a fault. Lastly, we can turn on/off specific motors (ID 20), shut down all motors (ID 100), and read the current voltage of the batteries (ID 30) as needed.

Commands need to be sent with the correct format for the firmware to recognize them. Each command sent from the Raspberry Pi to the Arduino is a string starting with the ID of the command (see Table 11.1 above). Any commands with arguments must have the string representation of any arguments in the same string separated by spaces. Finally, all commands must end with a new line character ("\n") to denote the end of the command. For example, a call to initialize all motors to their home positions (ID 1) will look like the following: “1\n”. A call to set motor ID 15 to move to position -34.5 degrees over a 1000ms period will look like the following: “10 15 -34.5 1000\n”. The firmware can easily parse this structure by reading all received bytes until the next newline character into a string. Then, every character before the first space can be interpreted as the command ID, and any remaining spaces in the string denote additional parameters. The firmware will attempt to parse the argument as either an integer (toInt()) or float (toFloat()) depending on the expected type of the parameter, but integers can also be parsed as valid floats using Arduino’s built-in toFloat() method.

Ava’s Arduino firmware is straightforward, essentially acting as a bridge between the Raspberry Pi and the HerkuleX motors. The firmware is the most commonly run program on the Arduino as it can run without direct human input when we are developing and testing Raspberry Pi code.

11.1.4 Debugging Scripts

We have written several Arduino debugging scripts to assist ourselves and future teams in both validating our firmware’s functionality and testing specific components when high level programs do not function as expected. We have written scripts for multiple levels of complexity to allow for testing multiple points of failure. In this section, we will talk about the most commonly used ones for development and testing.

Our primary tool for validating our electrical system and firmware is a script called Find_Addresses which checks to make sure power and communication is working for all motors. During setup, Find_Addresses reboots all motors and attempts to read the position of each motor we expect to be on the robot. The script also turns on greed LEDs for all connected motors, even ones with IDs we are not expecting. After initial setup is complete, the script repeatedly iterates through all 27 defined motors in Ava’s common Arduino library, attempting to read the position and error status of each. Motors that are powered and working properly will display their position, while ones that are off or disconnected will return an invalid angle and missing error status. The script also turns all found motor LEDs blue to indicate a successful command and response. Because the LEDs on each motor flash when they reboot, we can visually determine which motors are not receiving power when starting this script. Any connected motors with invalid IDs will appear green, and any connected motors that we expect to have will appear blue.

This script is versatile, allowing us to quickly check for motors without blue LEDs and also analyze the reported angles of expected motors for more detailed debugging.

To find motor home positions, we have a script called Go_Home_Pos. This script can iterate through each motor on Ava to read their angles one at a time, keep track of broken or disconnected motors, and send Ava to her home position. The script starts with Ava's lowest ID motor, prints the angle, and waits for user input in the serial window before moving on to the next motor. The script can also lock this motor in its angle before moving onto the next motor to incrementally hold Ava in her home position based on a boolean at the top of the file. This feature is useful when determining Ava's home position for a new motor, as we can move the motor by hand to its home position and record the offset in the common library. The script can also keep track of and log which motors have lost communication during the course of the program to aid in debugging intermittent wiring issues. If the corresponding boolean at the top of the file is set to true, the script will periodically check every expected motor and record the IDs of any motors which it cannot find. Lastly, the script can simply send all motors to their home positions to debug any visually incorrect offsets.

Lastly, we have HerkuleX_Readdress to change the ID of a given motor. Two parameters at the top of the file control the old ID and new ID, and the script will change any motor with the old ID to have the new ID. This is necessary when installing a fresh motor, as they all have IDs of 253 out of the box.

Our testing scripts have proved invaluable to debug both our circuitry and Arduino code. Not all issues with software are manifested by the code itself, so having basic checks for motor communication has been crucial for facilitating circuit testing and general fault detection.

11.2 Raspberry Pi Codebase

Like with the firmware, we overhauled the existing Raspberry Pi codebase to add several new features. First, we configured the Raspberry Pi to have an open ssh port for easy remote access with our standard development tools. We re-organized the code into a core set of classes which every script, in both simulation and on the real robot, could build off of. Lastly, we used our foundation to implement both locomotion scripts for complex tasks like walking and testing code for everything from individual methods/classes to entire movements.

11.2.1 Remote Connection to Raspberry Pi

Early on, we noticed a major roadblock when attempting to write code to run on Ava's Raspberry Pi. The Raspberry Pi lived inside the robot, meaning we could only directly use it by viewing the display through Ava's small LCD and connecting a USB mouse and keyboard to her head. This was unideal for rapid iteration of our Python code running on the Raspberry Pi during testing, so we sought to find a way to connect to the Raspberry Pi remotely.

The generally accepted method to connect remotely to a Raspberry Pi is through SSH, or Secure Shell Protocol. SSH allows one computer to connect to another remotely through a socket connection, meaning the connection is constantly held until stopped by either party. SSH is an industry standard method for remote connection, providing a secure, reliable, and easy method to do so. To set up the Raspberry Pi to allow SSH connections on WPI wifi, we registered our Raspberry Pi with the network, as normal with any computer or mobile device at WPI, and designated the default SSH port (22) to be open to SSH connections; this means anyone with the Raspberry Pi's hostname and password can connect remotely.

With SSH set up, we could now remotely connect to a headless terminal on the Raspberry Pi from any of our laptops. We also set up a VNC connection through a similar process to allow for a headed remote connection, meaning one could see the mouse and desktop rather than just typing into a terminal. For development purposes, we mainly connected to the Raspberry Pi via SSH through VS Code's "Remote - SSH" extension. This allowed us to open VS Code remotely from the Raspberry Pi, allowing us to write code with the same environment and features we were used to. This is significantly nicer than dealing with VNC's latency or terminal-based text editors' jank.

11.2.2 Code Structure

When refactoring code from last year, we took careful consideration in determining a sensical structure. Last year's codebase had some base classes, but most functionality was spread across various scripts for locomotion, simulation, and testing in separate folders. To create organized software that facilitated the swap from simulated to physical locomotion, we decided to move many things around. We have one package for all of the core robot classes and one package for scripts based on the core classes.

11.2.3 Core Classes

The core class package contains classes and methods corresponding to the physical components of the robot, like each motor, the IMU, and the Electromagnets. This package also includes motion control classes for kinematics, trajectory generation, and more. Because we aimed to implement the same locomotion functionality of unassisted standing and assisted walking in both simulation and real life, we decided to create a codebase that would allow us to easily switch between them. Since much of the code for theoretical control, like kinematics and trajectories, are referenced by both simulation and real classes, we decided to abstract as much of this functionality into separate classes.

The main class in this package is the Robot class, which represents the entire simulated or real robot. This class has one parameter in the constructor for whether the object is representing a real or simulated robot, and every method in the class changes its behavior depending on whether the robot is real or not. For instance, a real robot moves one of its motors by sending a serial command to its Arduino, while a simulated robot simply makes a CoppeliaSim library call. We considered using inheritance to make separate real and simulated robot classes, but we decided on having one consolidated class to make it even easier to change between simulation and physical testing in scripts. Instead of changing the instantiated class, one can simply change one boolean in a script. The robot class contains fields for each of its motors, sensors, and motion constants. It also contains many methods for initializing required components, kinematics calculations, and reading and setting motor positions.

The second most important class is the Motor class, which represents a real or simulated motor. Similar to the robot class, the Motor class also combines simulation and physical functionality into one set of methods. This class contains methods to read and set the position of one motor, which the Robot class references in its methods which actuate motors.

We have an IMU class for a simulated or real IMU, combined in the same way as the previous two classes. Our IMU is a wrapper class for either reading from the simulated IMU in CoppeliaSim or the real IMU on Ava, respectively. We also have implemented average and median filters which can be enabled to handle IMU noise within the wrapper, abstracted from scripts which reference IMU values.

For communicating with the Arduino on the physical robot, we have an ArduinoSerial class to construct commands in the correct format based on the command ID and arguments.

We have many more classes with parts of functionality, all of which work in unison to create the basis of a functional robot in both simulation and reality. To implement locomotion tasks such as walking and standing, we create Python scripts which make use of these basic classes.

11.2.4 Implemented Scripts

In our Testing package, we have a variety of scripts to either implement locomotion functionality or test various parts of our Python code. We will talk about the scripts responsible for our two major locomotion accomplishments, unassisted standing and assisted walking, in the next section. Here, we will cover how we construct basic locomotion scripts from our core classes and what testing scripts we have implemented to verify our code is robust and reliable.

Most locomotion scripts involve defining trajectories as variables, sending the motors to the locations specified at the start of the motion, and then repeatedly setting motor positions based on the desired trajectory. Our trajectory generation is written in MATLAB based on our work from previous Robotics Engineering courses, so we take the output of MATLAB scripts and paste the coefficients into our Python script. Once the robot is in the initial configuration, we start a loop and a timer for the duration of the motion. While the motion timer is not complete, we interpolate the trajectory at the time since we started looping to get the exact robot pose denoted at this time in the motion. Then, we command the motors to the desired position and continue with the next iteration of the loop. Interpolation math is very fast and allows for real-time trajectory following rather than attempting to keep up with a predefined set of via points. With this format, we can implement locomotion tasks by generating trajectories for the feet or hands to follow, copying them into Python, calculating joint angles designated at any given time through inverse kinematics, and commanding the motors to follow the trajectories.

We have also implemented several scripts to both serve as examples for future years and test basic parts of functionality. We have scripts to test the most basic functionality of our code such as sending and receiving commands from our Arduino on the real robot or reading measurements from our IMU. These scripts form the first level of validation to ensure that our most basic components, which everything else builds off of, function. Then, we have scripts to test basic primitive movements such as waving or handshaking. We have implemented and tested these hardcoded motions in Arduino, so we have a baseline for the latency and angles of these movements. These tests serve to verify our Raspberry Pi code and Arduino firmware are capable of performing the same primitive movements as our basic Arduino testing code. Finally, we have testing scripts to test entire motions such as leg trajectories. Isolating one trajectory is useful to check the specific trajectory waypoints without running an entire motion, generally consisting of multiple trajectories.

We have created many scripts for both implementing desired functionality and testing our code, which allowed us to be confident in both our base classes and locomotion scripts.

11.3 Standing and Walking

Our biggest accomplishments of this project are our locomotion scripts for unassisted standing and assisted walking. These represent the culmination of several months of circuitry iteration, simulation research, and software development. Our scripts went through many iterations of development and testing before arriving at our finalized versions.

11.3.1 Unassisted Standing Code

Our script for unassisted standing was created first and is less polished. It moves Ava from her home position to an upright standing position (see Figure 11.2) and attempts to actively balance her by keeping her center of mass over the support polygon formed by her feet.

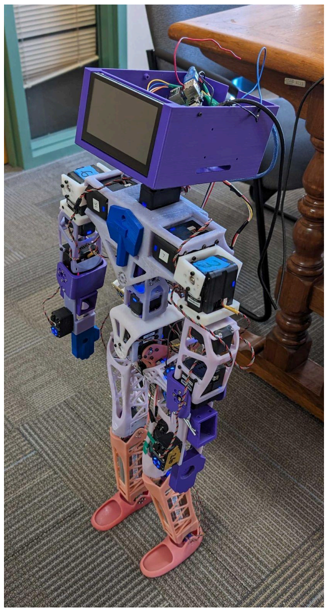

Figure 11.2: Ava Standing Unassisted

This script initializes a real Robot object and commands it to hold a relatively stable standing position. When she is in her initial standing position, she can remain upright for several minutes without any active counterbalancing. We start from this position to allow Ava to start by making minor adjustments rather than major motor movements.

Balancing consists of running two PID loops in parallel. One loop attempts to keep Ava's IMU level by moving her upper body forward or backward as needed, and the other loop attempts to keep her IMU level by moving her upper body from side to side as needed. We have several motors that can rotate her upper body about her hips in one and only one axis, which is useful to precisely control the specific location of her center of mass. After sending Ava to her starting position, the balancing code runs continuously to move her opposite to any direction she may be leaning.

11.3.2 Assisted Walking Code

After implementing unassisted standing on the physical robot, we aimed to implement functionality for Ava to walk with a cart. Walking assisted is the first step towards walking unassisted and also directly contributes to our overarching goal of creating assistive care humanoid robots. Her walking script consists of grabbing the cart and repeatedly taking steps with alternating feet.

First, we initialize a real Robot and command each motor to a position where she is leaning forward onto the top of the cart (see Figure 11.3).

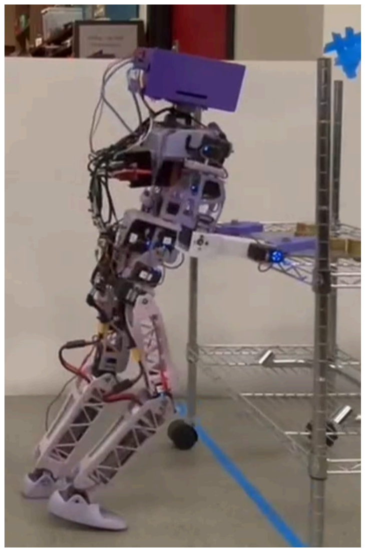

Figure 11.3: Ava Leaning Forward onto Her Cart, from Video [27]

From this position, we manually attach Ava to her cart using permanent magnets on her cart and a magnetic plate on top of the cart. The script waits for confirmation she is attached to the cart before executing the first trajectory. Once she starts walking, she follows alternating trajectories for each foot indefinitely. These trajectories run sequentially without any overlap in order for Ava to keep her balance. She will always have one foot on the ground to balance on. Like with other motion scripts, our trajectories were generated in MATLAB and copied over to Python. We have one trajectory for each foot, for a total of two trajectories. For each trajectory, we loop for the total trajectory time, interpolate the trajectory in real-time, and send the desired motor positions to the Arduino.

11.4 Conclusion

Over the course of this project, we have designed, integrated, and tested software across multiple codebases and multiple microcontrollers. While it was challenging to manage such large repositories working in parallel, we learned many important lessons about the importance of testing during iterative development. Starting with outdated code for a robot with different hardware, we built core functionality for generalized motions and then added scripts to perform complex locomotion tasks like standing and walking. We hope that the platform we have built will be useful for future years to implement more challenging motions such as unassisted walking or squatting on the physical robot.

In the next section, we will be discussing the results of our work on Ava's circuitry, simulation, and physical locomotion.

12. Results

In our work on Ava’s simulation, circuitry, and physical software, we have made several advancements to Ava’s capabilities. In this section, we will discuss in detail the results of our work in these three areas in terms of our successes, failures, and lessons learned.

12.1 Simulation Achievements

Throughout the academic year, we have not only met our requirements established in Chapter 4, but also exceeded them. By setting all of the joints to 0 degrees and running our balancing algorithm, the robot was able to stand straight and consistently for over 3.5 hours, after which we manually ended the simulation due to it not seeming to oscillate or leave the steady state. This was run 10 times, and successfully balanced for over an hour each of those times, giving us a success rate of 100%. As for the squatting motion, the robot performed squats for over an hour before we decided to manually end the simulation due to it not seeming to oscillate or leave the steady state. This was executed 10 times, and successfully performed squats for over 30 minutes 8 times, giving us a success rate of 80%. We found that these failures occurred due to compounding oscillation of the upper body when the balancing algorithm was not able to attenuate it.

Regarding assisted walking, in simulation, we set the floor size to 50 meters and the robot walked from end-to-end in about 50 minutes. The same balancing algorithm as described in Chapter 10.5 was used for all of these motions with the PID gains for actuation in the roll direction are 0.2, 0, and 0.1 respectively. The PID gains for actuation in the pitch direction are 0.25, 0, and 0.0075 respectively. This was tested 10 times, where the robot successfully walked over a meter within a minute 9 times, for a success rate of 90%. The failure occurred due to the robot not passing the 1 meter mark within 1 minute, which occurred due to the robot taking longer than usual to gain momentum with the cart. The motions of the right and left foot can be seen in Figure 12.1 and 12.2 respectively.

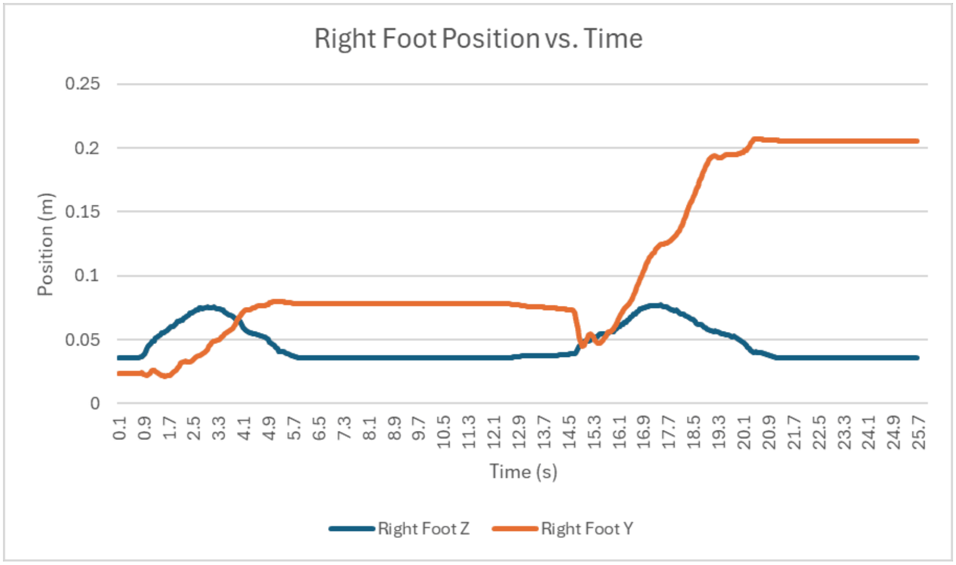

Figure 12.1: Graph of right foot's position over time while robot is walking forward.

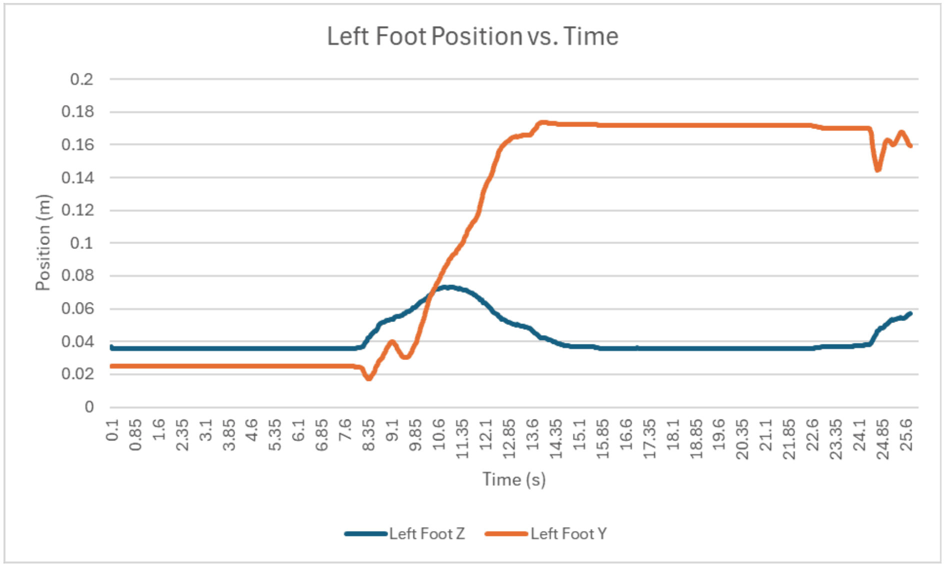

Figure 12.2: Graph of left foot's position over time while robot is walking forward.

12.2 Simulation Accuracy

A great benefit of having a simulated robot is to allow for testing of various algorithms without the risk of damaging the real robot, as mentioned previously. To evaluate the accuracy of the simulated robot's movements, we ran the same leg trajectory in 3 various environments. The first environment was within a MATLAB script, to show the ideal joint angles of the leg during its trajectory. The second environment was the simulation itself, graphing the legs joint angles in real-time as the leg moves throughout the trajectory. Both trajectories have the same via points and are calculated using the same methods.

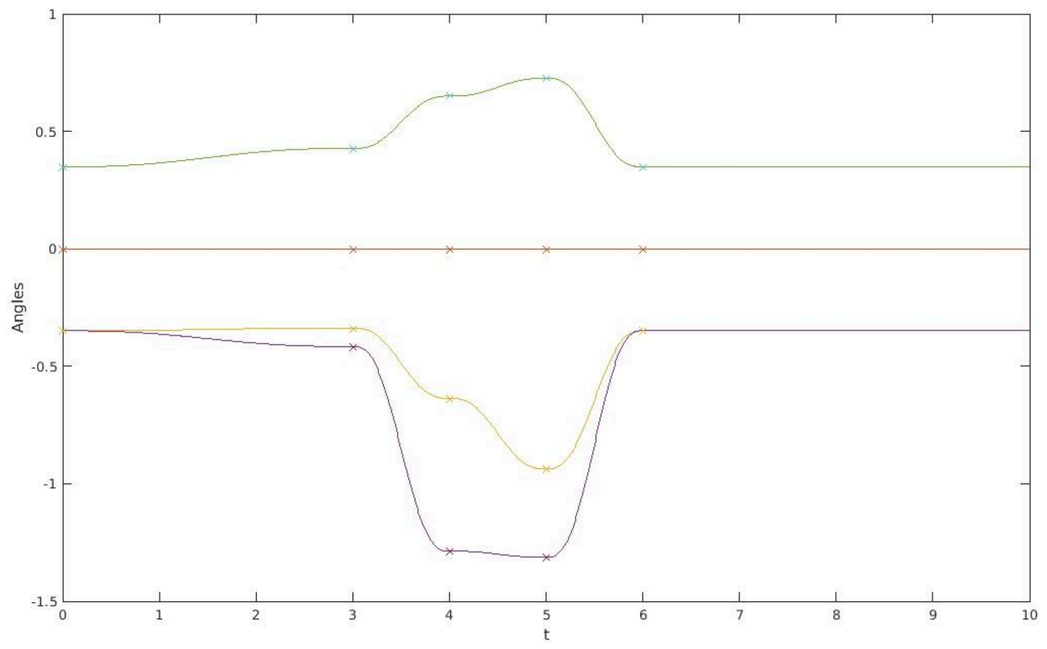

Figure 12.3: Graph of left leg's joint angles during walking script. The graph only depicts 1 cycle of Ava's gait.

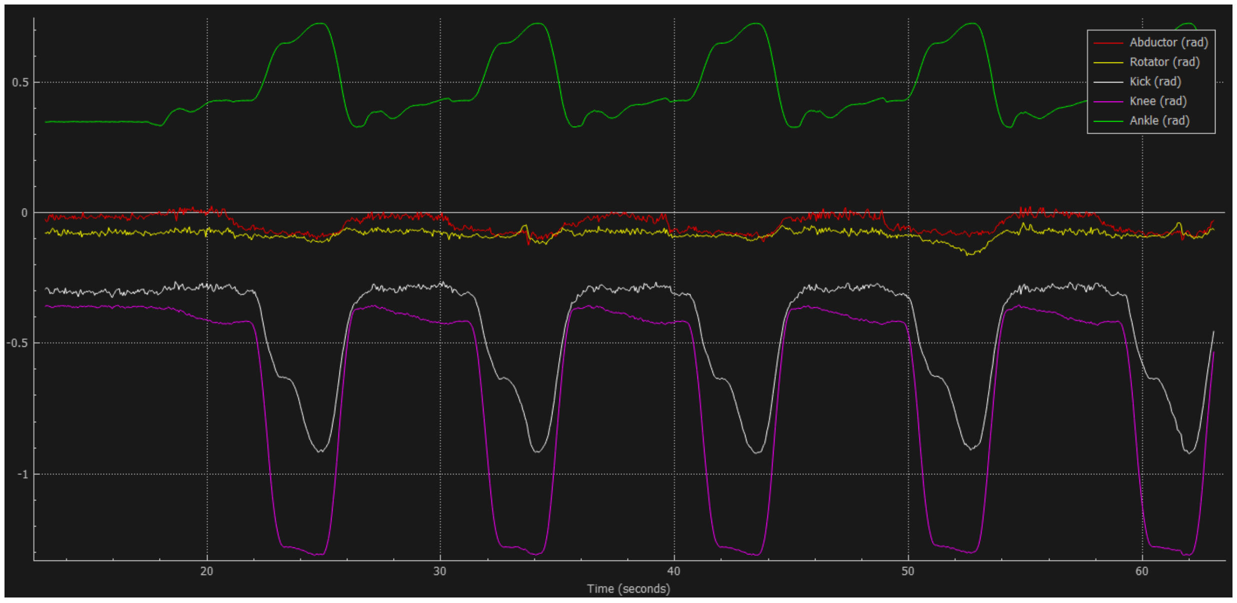

Figure 12.4: Graph of left leg's joint angles during walking script. Same trajectory as shown in Figure 12.3.

As shown in Figures 12.2 and 12.3, the trajectories of each joint within the simulation are similar to the ideal curves MATLAB calculated. Of course, there are more bumps in the simulation's graph and lines are not perfectly straight. This is due to the joints reacting to the dynamics of the system and experiencing disturbances during motion, such as contacting the ground or moving the mass of each link in the leg. These disturbances are not reflected in the MATLAB curve, thus creating a much smoother curve in MATLAB. The walking motion joint data, shown in Figure 12.4, was compared to the ideal, shown in Figure 12.3, and we found that the average error of each joint in radians was -0.0792, 0.1799, 0.0785, 0.0813, 0.0617 for the ankle, knee, kick, rotator, and abductor joints respectively. This falls within our acceptable range of 0.25 radians.

12.3 User Study Results

We recruited five people for the study. None of the participants had any issues setting up the simulation and getting our demo scripts to run, verifying that our instructions were sufficiently thorough. This resulted in us having a success rate of 100%. We primarily received positive comments on the clarification of the instructions. However, we received small criticisms on technical details that we resolved including not having prerequisite software downloads such as Python and Git, however they were able to figure out those additional steps without help. We then modified the instructions to include those additional steps and additional details we felt may be helpful.

12.4 Circuitry Achievements

When first designing our electrical system, we had three primary goals in mind: consistency, organization, and durability, as mentioned in Chaper 4. These should all come together to create an electrical system that can withstand one or more months of consistent use with no failures. After completion of our current circuitry iteration, we addressed each of the issues resulting in a well-rounded electrical system. We were able to achieve consistency through the implementation of our custom power and data distribution board in addition to the advent of our JST-XHP headers attached to our custom cable design. These solutions come together to allow for reliable and consistent electrical connection supplying power and data where it needs to be. Moreover, the power and data distribution board is significantly more organized; it provides a centralized location where the critical components of the system meet allowing for better cable management and ease of debugging. Most importantly, this system can achieve durability long term, primarily through the introduction of our custom cable design. The JST-XHP headers allow us to use only stranded wire in our design, creating a flexible wiring solution that is not prone to strain. In addition to this adding extra shrink-wrap tubing around particularly fragile parts of our cables allows for even further durability. These three solutions converge to establish the core of the robot, providing and distributing power and data throughout the system for over a month with no issues; successfully fulfilling our initial goals. We however, were also able to achieve an unforeseen goal, an easily alterable electrical system which allows for quick iteration. This, though not initially a design goal, turned out to be pivotal in the success of this project. The dynamic nature of this circuit and our software allows us to change motors types on a whim. This was critical when, towards the end of the project, we realized that the knee motors did not have sufficient torque. The system allowed us to effectively switch these motors to their higher torque counterparts with no changes to the system at all.

12.5 Physical Locomotion

Through extensive testing of our Arduino and Raspberry Pi code, we created a robust codebase capable of performing the same assisted walking motion from simulation on the physical robot. We also made significant progress with unassisted standing, but are not quite at the level of simulation yet.

For unassisted standing in simulation, we have found PID constants to have her balance for 3.5 hours with no signs of error. On the physical robot, we attempted to tune the PID constants multiple times to varying success. The longest unassisted stand we have achieved is 3 minutes. The main challenge with standing perfectly upright is mechanical slack in her links and joints. Even with all screws perfectly tightened and all motors locked, her center of mass can move a non-negligible amount due to compounding compliance across several links and joints in series. This makes the problem significantly more difficult, as it appears there is no single set of joint angles where she can perfectly balance without correction. Our PID constants either lead to massive overshoots with unpredictable movements or not enough corrective force to prevent her from falling over.

For assisted walking in real life, she was initially not able to support her weight during testing. Her knee motors experienced torque which exceeded the motor's acceptable limits, resulting in them automatically turning off due to their built-in failsafe protocol. To reduce the torque on these motors during assisted walking, we attempted to add springs between Ava's thigh and shin which would provide a force to assist the motor when her knees were bent. However, this did not meaningfully change the torque on the knee motors. It became clear the 0201 motors in the knees would not be capable of applying enough torque to hold up her upper body during walking. As a result, we replaced the 0201 knee motors with the stronger 0601 models with help from the design team for new thighs and shins, which resolved the issue.

We also modified the trajectories from the simulation motion after several iterations of differing foot heights and stride lengths. Our initial trajectories caused her feet to collide due to her legs not being far enough apart and her feet not moving far enough, and our final trajectories accounted for these issues. We also tested many iterations of initial motor positions to keep her leaning over the cart enough to push it but not too far as to fall over. With our current setup, Ava is pushing the cart entirely with the friction force between the magnets in her hands and the magnetic plate rather than any normal force. This is due to the way her wrists are oriented, as we did not have time to implement more human-like hands for a better grip on the cart. A wide variety of starting configurations and trajectories can work when walking in simulation, while working with the physical robot is less forgiving. Similar to unassisted standing, mechanical slack can cause Ava to lose balance and fall over easily, even while leaning on the cart. We meticulously planned our motions to ensure Ava remained as upright as possible throughout the entire walk.

With these changes, we were able to have Ava walk with her cart for over 2 meters. After this point, we were forced to stop her from walking into a desk because she showed no signs of stopping. A video of this motion is posted on Youtube [27]. We unfortunately did not have time to test the full extent of how far she can walk due to time constraints and concerns over parts breaking before a major demo. Prior to our investigation showing us which parts needed tightening, we ran eight trials of assisted walking, with only three of them resulting in Ava walking over a meter, which is a success rate of 37.5%. After reassembling and tightening much of the robot, we ran five trials of assisted walking, with four of those five resulting in her walking over 1 meter, which is a success rate of 80%.

12.6 Conclusion

Across all areas, we were able to achieve our goals for the project with reasonable success rates. We successfully created a circuit without major failures for over one month. We also developed simulation scripts to perform complex locomotion movements such as standing, walking, and squatting reliably. Finally, we modified the standing motion from the simulation to run on the physical robot for 3 minutes with limited success and modified the walking motion from the simulation to run on the physical robot for over 2 meters with great success.

In the next chapter, we will talk about the broader socio-economic implications of our results for the field of humanoid robotics and the world in general.

13. Discussion

13.1 Broader Implications

13.1.1 Economic Impact

Robotics as a field are generally very expensive as they require various motors, specialized components, and similar. This makes it so that only institutions that have large amounts of funding will be able to afford them, thereby increasing the quality of care, instruction, or whatever the purpose may be and improving relevant outcomes. Conversely, that means that in areas that are not as financially prosperous, there will be further disparities in outcomes. One of our primary goals for this project is to greatly reduce the overall cost of humanoid robots to increase accessibility. This will allow research groups, schools, hospitals, and whoever else may wish, to purchase the needed parts and construct one of our robots on their own, mitigating this disparity as much as possible. Additionally, by furthering the field of humanoid robotics this project coincidentally furthers the agendas of various companies and groups who wish to utilize humanoid robots. This increases the incentive to reduce costs and improve competition and accessibility further, hopefully decreasing the disparity in their usages.

13.1.2 Environmental Impact

Unfortunately, resin, PLA, and similar plastics used in the 3D-printing of our robot are toxic for the environment and hard to recycle. The usage of these materials needs to be done with care to avoid contamination in the environment. To address this, many parts of Ava were modified to shift and remove material as much as possible to reduce the amount needed without compromising on strength. By reducing the volume of plastics used in the construction of Ava, the environmental impact is lessened as less pollutants are released upon manufacturing, usage, and disposal. Also, the motors, fasteners, and similar are sourced from all over the world, causing pollution to be released upon shipping in addition to their manufacturing and disposal. For the future of this project, and for projects requiring ordering materials and parts, sourcing parts as locally as possible is ideal to mitigate this additional pollution. Overall, the manufacturing and usage of our robot will have a minimal impact on the environment.

13.1.3 Societal Influence

Humanoid robots are a very up-and-coming technology that the entire world has its eyes on. By producing a more cost-effective humanoid robot, we are not only contributing to the field in general but also making it such that more of these robots will be seen in the future. The uses of humanoid robots are as varied as the tasks humans themselves can perform and so it is important to consider whether introducing these robots into certain fields will have a positive or negative effect on the people in those fields and adjacent ones. While it is very likely that in the future many jobs will be replaced with robots, which will greatly impact especially lower-income and blue-collar workers. To mitigate this, it is important to have job training accessible so these workers can shift into roles that can utilize their skills while also managing robots working with them. It will become a difficult question for employers to keep human workers or replace them with free robotic labor, but at least for the time being it will be extremely important to have people who are familiar with the robots to work alongside them.

13.1.4 Political Ramifications

While robotics is becoming much more prominent and accepted in many technologically advanced countries, it is important to consider how humanoid robotics may influence the culture of other countries and the global markets we are all a part of. As humanoid robots become more commonplace, socializing with them will likely become much more common and accepted, causing them to integrate further into our lives, media, and culture as a whole. Especially with artificial intelligence being developed alongside humanoid robotics, legislation will need to be considered regarding how realistic and in what capacities these robots are allowed to be deployed. There are a lot of benefits of robots that are indistinguishable from people, as they can do anything that a person can do, but should they be treated the same as people? If human labor becomes economically unreasonable due to not needing to pay robots, there are lots of potential social hazards that may occur because of this. While we are nowhere near being able to develop anything like this, it is a very serious concern that will need to be investigated.

13.1.5 Ethical Concerns

There are many ethical concerns regarding robotics and automation, but these concerns become much more serious with humanoid robotics. Humanoid robots are defined by their capacity to do similar work to humans, and this will only become more glaring as they become more advanced. While our project is not advancing humanoid robotics into that sphere, there are many concerns regarding these robots taking over jobs, hurting people, and eventually potentially even being indistinguishable from people. Similar to the concerns of societal influence and political ramifications, there is great ethical concern about the development of humanoid robotics. To mitigate these concerns, legislation will need to be passed while keeping the lives of everyday people in mind to ensure the economy does not evolve into one where any besides those with the highest degrees and most specific skills can thrive.

13.1.6 Health and Safety Issues

The primary health and safety concern regarding this project is with the materials themselves. Resin and other 3D-printing materials are toxic and require proper training and protective equipment to use safely. If proper precautions are not taken, respiratory and similar issues may occur. This is most notable with 3D-printing with resin as the liquid resin is very volatile and the fumes, when inhaled, can lead to respiratory issues. To mitigate this, wearing proper respirator masks is extremely important.

13.1.7 Manufacturability

A major component of this project is making it as easy as possible to replicate the robot constructed. As this is a fairly complex robot, with dozens of different parts and hundreds of fasteners, it can be very difficult to assemble it properly. Additionally, it is often difficult to get specific parts to construct robots. To mitigate this, all of the part files are available and open-source for 3D-printing, along with all other components being easy to purchase and inexpensive as possible. Additionally, many parts have been reoriented and redesigned such that assembling and debugging the robot is simple. The software, firmware, and circuitry is all available and described in detail such that their reproduction, use, and customizability are all as easy as possible.

13.2 Engineering Standards

There are many engineering standards that can be applied to this project, including those regarding the usage of robotics, circuitry in medical devices, software architecture, and more. Before discussing different safety and similar requirements, ISO 8373 defines what a robot actually is and therefore what all of the following standards pertain to. There are several standards pertaining to designing and using robotics in the medical field. One of the most important ISO standards for this project is relating to safety. ISO 13482 discusses the safety requirements for robots in healthcare, specifically with personal care robots. ISO 20218-1 discusses how to design safe and effective end-effectors for the robot, which will be extremely important as the end-effector is how the robots interact with the world. Similarly, SA - P7017 discusses considerations for the design and methods of assistive robots. ISO 23482-1 will be extremely important if our project is to enter the medical field as it expands upon ISO 13482 by stating different methods to test the safety requirements listed. There are also several ISO standards regarding nomenclature including ISO 9787, which defines and specifies robotic coordinate systems including notation and common nomenclature. These is very relevant to this project, as we perform lots of frame assignments throughout our kinematics. As robotics are further integrated into the workforce, it is important to specify how they are allocated into an organization's framework and how to document and improve this. That is documented by ISO 30434 and SA - P3107. As the robot has many electrical components that may pose potential risks, it is important to consider IEC 60601, which provides guidance on medical electrical equipment that is in some way autonomous. Similarly IEC 80601-2-78 relates these electrical systems more closely to robotics by giving requirements for basic safety and performance with regards to the robot. Given that our robot is 3D-printed, ISO 52939 may also be considered, as it discusses different structural and infrastructural elements of additive manufacturing. Finally, the software itself needs to follow relevant standards as well, such as ISO 42030 which goes over how to organize and record architecture evaluations, systems, and similar to allow for validation and verification of software systems.

This chapter described several potential broader implications of this project on health, society, ethics, and similar. It also goes over various engineering standards that are potentially applicable to this project. The next chapter discusses the conclusions drawn based on all of the work completed, along with the results and discussions of the work and future recommendations.

14. Conclusions

Considering the project as a whole, we have met and in some areas exceeded the goals we had set out to accomplish. There are three main categories for our project them being the electrical system, the simulation, and the software/locomotion of the real robot. While each of these areas are distinct in their own right, they come together to create a cohesive system which works together to further our research into humanoid robotics. We will discuss in more detail the conclusions realized in each of these categories.

14.1 Simulation Conclusions

Firstly, we will discuss simulation, we have successfully reached our target goals established at the beginning of the year. Firstly, our team increased the accuracy of the simulated model of Ava within CoppeliaSim to enable more in-depth algorithm testing. Additionally, we created multiple scripts to demonstrate various feats for locomotion within simulation, such as standing without assistance, squatting without assistance, and walking with the assistance of a simulated cart. To perform these feats, kinematic calculations were done to establish the relationship between the robot's joint space versus its task space. This includes calculations for forward, inverse, and differential kinematics.

14.2 Circuitry Conclusions

As for circuitry, our attempt to engineer a resilient electrical system for our humanoid robot has resulted in significant advancements. With a steadfast focus on consistency, organization, and durability, we integrated custom power and data distribution boards, complemented by JST-XHP headers and a custom cable design. These solutions unify to ensure dependable power and data transmission, thereby enhancing performance and simplifying troubleshooting.

Moreover, our commitment to durability has proven itself valuable in strengthening the system against the rigors of continuous operation. By incorporating stranded wire and additional shrink wrap tubing, we have effectively mitigated the risk of strain-induced failures, bolstering the system's resilience over prolonged periods.

The culmination of these endeavors has met our initial objective of sustaining functionality for over a month without encountering any failures within the electrical system. This achievement marks a significant milestone in our robot's development, providing a robust foundation for future advancements and innovations.

14.3 Software and Locomotion Conclusions

In addition to overcoming the challenges of wrestling with circuitry issues and starting from scratch in simulation, we also created code to allow Ava to stand and walk in the real world. This massive accomplishment is the result of many hours of iteration and rigorous testing, and we are extremely proud of our progress. We not only developed robust codebases in Arduino and Python which future teams can build upon, but also built upon our own framework to achieve complex locomotion tasks. By gradually developing and testing our code throughout this project rather than rushing to write working code quickly, we were able to achieve our goals for the physical robot code and more.

14.4 Recommendations for Future Work

14.4.1 Impedance Control

During our testing of the real robot, we used simple position control to actuate each joint within the robot. Although position control is very simple, issues can arise when the manipulator is trying to reach a position that is being blocked by an external disturbance (the arm pushing against a wall for example). This behavior can risk damage to the servo motors actuating the joints, as they would experience an overload during this movement. Impedance control is a type of control using joint torques as an input to reach a desired end effector position. The benefit of impedance control is that the manipulator will move with an external force if desired, or can heavily dampen translation and/or rotation in certain axes. Impedance control can improve the health of Ava’s joints during locomotion since you would be controlling their output torque. This can also benefit human-robot interaction, as the forces exerted by a human will not strongly affect the robot. The reverse is also true, where a robot using impedance control would not risk harming a human via strong movements.

14.4.2 Implement Zero Moment Point Tracking

A goal that our team wanted to achieve was unassisted walking on the real robot and in simulation. A large obstacle was fully understanding and implementing the control of the robot's zero-moment point. Our team failed to implement this control due to time constraints. Once our team felt confident enough to begin researching and implementing ZMP control, it was already halfway through C term. Thus, we decided to focus on ensuring assisted walking worked reliably. With stronger knowledge of ZMPs and higher quality simulation scripts, we hope that unassisted walking via ZMP tracking will be possible next year.

14.4.3 Compare Simulation Results to Real Robot Results

By the end of the academic year, we were able to generate and plot data regarding Ava's left leg's joint angles during walking trajectories from both CoppeliaSim and an ideal curve calculated from MATLAB. The next step would be to compare these data points with the joint angles the real robot travels through during Ava's walking script. This comparison can truly determine the simulation's accuracy to the real world via quantitative data.

References

[1] Poppy. “History.” *Poppy Project*, 2016, poppy-project.org/en/about.

[2] Galgano, Anthony, Fournet, David, Lehman, Alexandria, Engdahl, William and Beazley, Raymond. *3D Printed Humanoid Robot Project.* (Undergraduate Major Qualifying Project No. E-project-042822-123507) Retrieved from Worcester Polytechnic Institute Electronic Projects Collection: https://digitalwpi.wpi.edu/concern/student_works/dr26z168n.

[3] Fernandez, Joshua, Snow, Casey, O’Sullivan, Finbar and Lee, Erin. *Simulation and Testing of Stabilization and Assisted Walking in a 3D-Printed Humanoid Robot.* (Undergraduate Major Qualifying Project No. E-project-050323-153614 108536) Retrieved from Worcester Polytechnic Institute Electronic Projects Collection: https://digital.wpi.edu/show/hd76s353h.

[4] The Poppy Project. “OPEN SOURCE PLATFORM FOR THE CREATION, USE AND SHARING OF INTERACTIVE 3D PRINTED ROBOTS.” (n.d.). URL https://www.poppy-project.org/en/.

[5] Team, Robots. “Asimo.” ROBOTS, 18 May 2018, robotsguide.com/robots/asimo.

[6] Poppy. “History.” Poppy Project, 2016, poppy-project.org/en/about.

[7] Lytle, Tessa, Austin, Emily, Milligen, James Van, Alarcon, Zenia and Alag, Aashish Singh. *Development of 3D Printed Humanoid Robots* (Undergraduate Major Qualifying Project No. E-project-050323-145305 108466) Retrieved from Worcester Polytechnic Institute Electronic Projects Collection: https://digital.wpi.edu/concern/student_works/br86b7141?locale=es.

[8] Protokinetics. “Understanding Phases of the Gait Cycle.” (n.d.)., URL https://protokinetics.com/understanding-phases-of-the-gait-cycle.

[9] Lynch, M. Kevin and Park, Frank C. “Modern Robotics.” (2024). URL https://hades.mech.northwestern.edu/index.php/Modern_Robotics.

[10] Amazon Web Services. “What is XML?” (n.d.). URL https://aws.amazon.com/what-is/xml/.

[11] Formant. “URDF.” (n.d.), URL https://formant.io/urdf/.

[12] Farley, Andrew, Wang, Jie, Marshall, Joshua A. "How to pick a mobile robot simulator: A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion" Simulation Modelling Practice and Theory (2022).

[13] Ayala, Angel, Cruz, Francisco, Camps, Diego, Rubio, Rodrigo, Fernandes, Bruno, Dazeley, Richard. "A Comparison of Humanoid Robot Simulators: A Quantitative Approach" ICDL-EpiRob (2020).

[14] Dekker, M.H.P. "Zero Moment Point Method For Stable Biped Walking." Technical report no. Eindhoven, University of Technology. 2009.

[15] 3D Systems. "What is an STL File?" (n.d). URL https://www.3dsystems.com/quickparts/learning-center/what-is-stl-file.

[16] ROS. "SolidWorks to URDF Exporter." (2021). URL https://wiki.ros.org/sw_urdf_exporter.

[17] CoppeliaSim ((n.d.)). URL https://coppeliarobotics.com/.

[18] CoppeliaSim. "CoppeliaSim User Manual." (n.d). URL https://manual.coppeliarobotics.com/.

[19] Generation Robots. "Poppy Humanoid Robot (with 3D parts)." (n.d.). Accessed March 18, 2024, URL https://www.generationrobots.com/en/403347-poppy-humanoid-robot-raspberry-pi-version-with-3d-parts.html.

[20] The Poppy Project. "Poppy project documentation." (n.d.). Accessed March 18, 2024, URL https://docs.poppy-project.org/en/.

[21] Aldebaran Robotics. "NAO Documentation." (2022). Accessed March 18, 2024, URL http://doc.aldebaran.com/2-8/home_nao.html.

[22] RobotLAB. "NAO Power V6 Standard Edition." (n.d.). Accessed March 18, 2024, URL NAO Robot Power V6 Standard Edition for Research (robotlab.com).

[23] "Robocup Federation Official Website." *RoboCup Federation Official Website*, 4 July 2023, www.robocup.org/

[24] PlatformIO ((n.d.)). URL https://platformio.org/.

[25] Dongbu Robot. "HerkuleX DRS-0101/DRS-0201 User Manual." (2011) URL https://data2.manualslib.com/pdf7/218/21729/2172847-dongbu_robot/herkulex_drs0101.pdf

[26] Dongbu Robot. “HerkuleX DRS-0601 User’s Manual.” (2015) URL https://www.sgbotic.com/products/datasheets/robotics/Herkulex0601%20manual.pdf.

[27] Youtube. “Ava Walking Assisted with Cart for 2 Meters (2024).” (2024) URL https://youtu.be/RQEbH795WU4

Appendix A

General Configuration Information

Motor Name, Defined within CoppeliaSim	**Joint Twist**	**Link Home Configuration to Center of Mass**	**Controlled Link Home Configuration**	**Controlled Link Mass (g)**
Right Shoulder Rotator	[-1, 0, 0, 0, -76, 73]	[[1, 0, 0, -93.1], [0, 1, 0, 73], [0, 0, 1, 76], [0, 0, 0, 1]]	[[1, 0, 0, -101.09], [0, 1, 0, 74.53], [0, 0, 1, 67.02], [1, 0, 0, 1]]	10.17
Right Shoulder Abductor	[0, 0, 1, -73.05, -119.1, 0]	[[1, 0, 0, -117.6], [0, 1, 0, 73], [0, 0, 1, 90.5], [0, 0, 0, 1]]	[[1, 0, 0, -127.29], [0, 1, 0, 72.44], [0, 0, 1, 62.05], [1, 0, 0, 1]]	71.23
Right Bicep	[1, 0, 0, 0, 76.65, -72]	[[1, 0, 0, -165.5], [0, 1, 0, 73], [0, 0, 1, 64.4], [0, 0, 0, 1]]	[[1, 0, 0, -229.68], [0, 1, 0, 71.22], [0, 0, 1, 67.37], [0, 0, 0, 1]]	206.87
Right Elbow	[0, 1, 0, -64.63, 0, -247.17]	[[1, 0, 0, -274.38], [0, 1, 0, 98.9], [0, 0, 1, 51.47], [0, 0, 0, 1]]	[[1, 0, 0, -330.19], [0,1,0,73.82], [0,0,1,54.34], [0,0,0,1]]	235.84

Motor Name, Defined within CoppeliaSim	**Joint Twist**	**Link Home Configuration to Center of Mass**	**Controlled Link Home Configuration**	**Controlled Link Mass (g)**
Right Wrist	[0, -1, 0, 53.02, 0, -383.96]	[[1, 0, 0, -384.13], [0, 1, 0, 85.8], [0, 0, 1, 45.49], [0, 0, 0, 1]]	[[1, 0, 0, -436.72], [0, 1, 0, 61.17], [0, 0, 1, 45.68], [0, 0, 0, 1]]	50.04
Left Shoulder Rotator	[1, 0, 0, 0, 76, -73]	[[1, 0, 0, 93.1], [0, 1, 0, 73], [0, 0, 1, 76], [0, 0, 0, 1]]	[[1, 0, 0, 101.09], [0, 1, 0, 74.53], [0, 0, 1, 67.02], [1, 0, 0, 1]]	10.17
Left Shoulder Abductor	[0, 0, 1, -73.05, 119.1, 0]	[[1, 0, 0, 117.6], [0, 1, 0, 73], [0, 0, 1, 90.5], [0, 0, 0, 1]]	[[1, 0, 0, 127.29], [0, 1, 0, 72.44], [0, 0, 1, 62.05], [1, 0, 0, 1]]	71.23
Left Bicep	[-1, 0, 0, 0, -76.65, 72]	[[1, 0, 0, 165.5], [0, 1, 0, 73], [0, 0, 1, 64.4], [0, 0, 0, 1]]	[[1, 0, 0, 229.68], [0, 1, 0, 71.22], [0, 0, 1, 67.37], [0, 0, 0, 1]]	206.87
Left Elbow	[0, 1, 0, -64.63, 0, 247.17]	[[1, 0, 0, 274.38], [0, 1, 0, 98.9], [0, 0, 1, 51.47], [0, 0, 0,	[[1, 0, 0, 330.19], [0,1,0,73.82], [0,0,1,54.34],	235.84

Motor Name, Defined within CoppeliaSim	**Joint Twist**	**Link Home Configuration to Center of Mass**	**Controlled Link Home Configuration**	**Controlled Link Mass (g)**
		1]]	[0,0,0,1]]	
Left Wrist	[0, -1, 0, 53.02, 0, -383.96]	[[1, 0, 0, 384.13], [0, 1, 0, 85.8], [0, 0, 1, 45.49], [0, 0, 0, 1]]	[[1,0,0,436.72], [0,1,0,61.17], [0,0,1,45.68], [0,0,0,1]]	50.04
Hips Front to Back	[-1, 0, 0, 0, -49.9, -114.02]	[[1, 0, 0, -26.65], [0, 1, 0, -130.09], [0, 0, 1, 55.73], [0, 0, 0, 1]]	[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]	0
Torse Side to Side	[0, 0, -1, 0, 0, 0]	[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 5.63], [0, 0, 0, 1]]	[[1,0,0,1.65], [0,1,0,3.44], [0,0,1,40.50], [0,0,0,1]]	0.13
Hips Rotate	[0, -1, 0, 59.53, 0, -1.1]	[[1, 0, 0, 1.1], [0, 1, 0, -73.1], [0, 0, 1, 59.53], [0, 0, 0, 1]]	[[1,0,0,1.65], [0,1,0,3.44], [0,0,1,40.50], [0,0,0,1]]	129.13
Hips Side to Side	[0, 0, -1, -130.05, 1.2, 0]	[[1, 0, 0, 1.1], [0, 1, 0, -130.09], [0, 0, 1, -13.97], [0, 0, 0, 1]]	[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]	258.41

Motor Name, Defined within CoppeliaSim	Joint Twist	Link Home Configuration to Center of Mass	Controlled Link Home Configuration	Controlled Link Mass (g)
Torso Front to Back	[0, -1, 0, 58.47, 0, -1.2]	[[1, 0, 0, -19.6], [0, 1, 0, 0], [0, 0, 1, 60.53], [0, 0, 0, 1]]	[[1,0,0,0.6], [0,1,0,-85.27], [0,0,1,33.17], [0,0,0,1]]	47.13
Right Leg Abductor	[0, 0, 1, -191.08, 21.44, 0]	[[1, 0, 0, -23.64], [0, 1, 0, -191.09], [0, 0, 1, 57.93], [0, 0, 0, 1]]	[[1,0,0,-21.44], [0,1,0,-191.08], [0,0,1,57.93], [0,0,0,1]]	74.77
Right Leg Rotator	[0, -1, 0, 45.03, 0, 65.44]	[[1, 0, 0, -67.64], [0, 1, 0, -188.39], [0, 0, 1, 45.03], [0, 0, 0, 1]]	[[1,0,0,-65.44], [0,1,0,-188.39], [0,0,1,45.03], [0,0,0,1]]	129.77
Right Kick	[-1, 0, 0, 0, -41.53, -209.39]	[[1, 0, 0, -94.54], [0, 1, 0, -209.39], [0, 0, 1, 41.53], [0, 0, 0, 1]]	[[1,0,0,-91.34], [0,1,0,-209.39], [0,0,1,41.53], [0,0,0,1]]	109.13
Right Knee	[-1, 0, 0, 0, -41.53, -393.29]	[[1, 0, 0, -62.33], [0, 1, 0, -393.29], [0, 0, 1, 41.53], [0, 0, 0, 1]]	[[1,0,0,-60.13], [0,1,0,-393.29], [0,0,1,41.53], [0,0,0,1]]	512.23

Motor Name, Defined within CoppeliaSim	**Joint Twist**	**Link Home Configuration to Center of Mass**	**Controlled Link Home Configuration**	**Controlled Link Mass (g)**
Right Ankle	[1, 0, 0, 0, 41.53, 624.84]	[[1, 0, 0, -22.08], [0, 1, 0, -624.84], [0, 0, 1, 41.53], [0, 0, 0, 1]]	[[1,0,0,-19.88], [0,1,0,-624.84], [0,0,1,41.53], [0,0,0,1]]	57.91
Left Leg Abductor	[0, 0, 1, -191.09, -23.64, 0]	[[1, 0, 0, 23.64], [0, 1, 0, -191.09], [0, 0, 1, 57.93], [0, 0, 0, 1]]	[[1,0,0,23.64], [0,1,0,-191.09], [0,0,1,57.93], [0,0,0,1]]	74.77
Left Leg Rotator	[0, -1, 0, 45.03, 0, -67.64]	[[1, 0, 0, 67.64], [0, 1, 0, -188.39], [0, 0, 1, 45.03], [0, 0, 0, 1]]	[[1,0,0,67.64], [0,1,0,-188.39], [0,0,1,45.03], [0,0,0,1]]	129.77
Left Kick	[1, 0, 0, 0, 41.53, 209.39]	[[1, 0, 0, 94.54], [0, 1, 0, -209.39], [0, 0, 1, 41.53], [0, 0, 0, 1]]	[[1,0,0,93.54], [0,1,0,-209.39], [0,0,1,41.53], [0,0,0,1]]	103.84
Left Knee	[1, 0, 0, 0, 41.53, 393.29]	[[1, 0, 0, 62.33], [0, 1, 0, -393.29], [0, 0, 1, 41.53], [0, 0,	[[1,0,0,62.33], [0,1,0,-393.29], [0,0,1,41.53], [0,0,0,1]]	512.23

Motor Name, Defined within CoppeliaSim	Joint Twist	Link Home Configuration to Center of Mass	Controlled Link Home Configuration	Controlled Link Mass (g)
		0, 1]]		
Left Ankle	[-1, 0, 0, 0, -41.53, -624.84]	[[1, 0, 0, 22.08], [0, 1, 0, -624.84], [0, 0, 1, 41.53], [0, 0, 0, 1]]	[[1,0,0,22.08], [0,1,0,-624.84], [0,0,1,41.53], [0,0,0,1]]	57.92
Head Nod	[1, 0, 0, 0, 32.2, -144.1]	[[1, 0, 0, 17.8], [0, 1, 0, 144.1], [0, 0, 1, 32.2], [0, 0, 0, 1]]	[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]	0
Neck	[0, 1, 0, -32.2, 0, 0]	[[1, 0, 0, 0], [0, 1, 0, 124.1], [0, 0, 1, 32.2], [0, 0, 0, 1]]	[[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]	0

Appendix B: GitHub Organization

The following GitHub organization contains public repositories with software developed over the past three years of this project.
https://github.com/KoalbyMQP
The code we have written as part of this year's MQP can be found in the following repositories:
https://github.com/KoalbyMQP/Arduino-Code_23-24
https://github.com/KoalbyMQP/RaspberryPi-Code_23-24

Reflections

The completion of this project required the collaboration of all team members with their unique specialties. This team consisted of different members specializing in software development, controls, circuitry, mechanics, kinematics, dynamics, fabrication, and more. Through the delegation into subteams and various tasks by specialty, we were able to streamline the continued reworking and development of this project with efficiency and success. Strong communication between individual members, subteams, the locomotion team, and the design team was critical to our success by maximizing our efficiency and tackling problems from all sides.

Through working on this project, we all further developed our technical, organizational, and collaborative skills. Given that this project was completed over the entire academic year, we gained invaluable experience working on large-scale engineering projects that we would not have working on shorter-term projects.

This project successfully continued the work of the previous 3D Printed Humanoid Robotics MQP teams by overhauling the simulation to work with the updated CAD models, implemented unassisted standing and basic balancing tasks in simulation and reworked the electronics of the physical robot to increase safety, robustness, and functionality.

Scott Pena

My role throughout this academic year was to further the accuracy and effectiveness of the simulation of Ava through CoppeliaSim. In the beginning of A-Term, I had no experience working in CoppeliaSim. The only experience in simulation was through Gazebo from RBE3002. As I was getting familiar with CoppeliaSim, I worked on recalculating the kinematics of Ava. I began to use Denavit-Hartenberg parameters through my experience from RBE3001 and RBE500, but I opted to use concepts around joint twists for kinematics from my learning in RBE501. Using these kinematics, I also created the joint trajectories for locomotion during walking using knowledge from RBE3001. Once I was more familiar with CoppeliaSim, I began to use my SolidWorks skills from RBE2001 to create a URDF of the robot to import Ava's 3D geometry into CoppeliaSim.

Ana Roure

Throughout this project, my main responsibilities revolved around installing the Raspberry Pi, IMU, and handling 3D printing and part replacement tasks. Initially, I installed the Raspberry Pi into the robots, followed by setting up and testing the IMU to ensure its accuracy for standing unassisted testing. Additionally, I played a crucial role in replacing parts on Ava whenever they broke down. My proficiency with 3D printers proved to be extremely helpful, as I regularly worked with multiple printers each week to swap out old parts for new ones. Furthermore, my background as an RBE major equipped me with extensive experience in wiring and electrical engineering. Consequently, tasks such as soldering and connecting wires were familiar territory for me as I worked on recreating the wiring diagram to real life.

Dylan Nguyen

My responsibilities in this project have been multiple. Primarily I focused on the electrical system alongside my esteemed colleague Stephen Fanning. I worked on the previously mentioned goals of circuitry consistency, organization, and durability. My time as a robotics engineering student at WPI had set me up well with proper debugging practices as well as the skills to create a functional circuit for our robot. Using an iterative process for circuitry was paramount in our success and in my development process. Moreover, I worked on some software implementation in the robot, deploying a median filter for our IMU, tuning our PID balancing, and converting our previous Arduino codebase into PlatfromIO for ease of use. This implementation is informed by what I have learned as a Computer Science student at WPI. Finally, I served as our de facto Project Manager employing the skills I learned in the Software Engineering class to keep our tasks well organized and to ensure all members of our team had sufficient tasks every week of our project.

Jatin Kohli

My primary role has been developing, testing, and maintaining the locomotion software for the physical robot, Ava. I have created and organized the GitHub repositories for this year's code by managing various branches and documentation. I have significantly contributed to both the Arduino and Raspberry Pi codebases, helping to design and implement the Arduino firmware and Python classes responsible for the ability of the Raspberry Pi to command the HerkuleX motors through the Arduino. I refactored the Find_Addresses script from last year to more accurately find motors, re-wrote the firmware to support a new serial command structure, and found motor-home positions more times than I can count. I also wrote a large portion of the Robot, Motor, and IMU classes which are used in both physical and simulated locomotion code to control the robots. I have also helped write many test scripts for both codebases, ensuring our code is correct and reliable amidst non-ideal conditions of mobile circuitry and destructible 3D-printed parts. As a Computer Science and Robotics Engineering Major, I have gained experience with writing and testing industry-level software for an extremely complex robotic system. Managing code running across multiple microcontrollers to interface with sensors and actuators has proved challenging yet valuable, and I feel confident that the skills I have gained from this project will carry over into my work and life after WPI.

Jack Rothenberg

Throughout this project, my roles have primarily been furthering the software and simulation of Ava. In addition to the specific focuses I took throughout the year, I also worked extensively on various hardware components, 3D-printing, circuitry, and helping wherever I could be of use. In A-term I was primarily working on the hardware of the robot to get her back in one piece and also beginning working on the software to rework it such that it can be used with the simulation. Also began lots of work in background research on balancing and trajectory generation. My work in RBE 3001, RBE 500, BME 580, and RBE 3002 were all extremely helpful for this, as RBE 3001, RBE 500, and BME 580 gave me a strong background in trajectories, assigning frames, and how to think about manipulators mathematically. This was all extremely useful as I worked on the software for trajectory generation and for the robot as a whole. My work in RBE3002 and RBE500 was very helpful as I transitioned into working on the simulation in CoppeliaSim, as they both involved developing in robotics simulators. Throughout the rest of the year, I was working on the software and simulation side, ensuring that all of the code written to run on the simulated robot could also run on the real robot. Good coding practices along with lots of object-oriented concepts that I learned in CS 3733 and CS 2102 were vital to this project as the organization through class structures allowed us to modularly create and set methods to be used interchangeably whether running on the real or simulated robot.

Throughout the entire project, I had to read many research papers on human walking gait, robotics, and many other topics related to the project. So many of my BME classes, including BME 3300, BME 4503, and BME 4023, helped me develop the skills to find, analyze, critique, and utilize academic literature, all skills that were invaluable to tackling such a difficult problem. BME 3300 especially gave me experience working on a group research project involving lots of literature review in which having a strong understanding of the literature being read is vital.

Stephen Fanning

I had the privilege of working on a diverse range of tasks for this project, ranging from high-level trajectory control to low-level serial communication, but by far my largest contribution to this project was the circuit. Throughout A, B, and C terms, Dylan and I spent most of our time dedicated to the surprisingly difficult task of making the robot consistently turn on and move. As a result of our work, Ava now has a consistent, robust, and flexible circuit, and this consistency should significantly help the future goals of this project. I was then able to help overhaul the firmware and refactor, develop, and adjust our walking trajectories. I hope that, as a result of this work, future teams will have an excellent foundation to build on, and they will be able to achieve even greater feats than we were able to ourselves.