



Simulation and Testing of Stabilization and Assisted Walking in a 3D-Printed Humanoid Robot

IN PARTIAL FULFILLMENT OF A MAJOR QUALIFYING PROJECT
WORCESTER POLYTECHNIC INSTITUTE

Submitted by:

Joshua Fernandez (RBE)

Erin Lee (RBE/ME)

Tessa Lytle (ME)

Finbar O'Sullivan (ME)

Casey Snow (CS)

Project Advisors:

Kaveh Pahlavan (ECE/CS)

Pradeep Radhakrishnan (ME/RBE)

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see <http://www.wpi.edu/academics/ugradstudies/project-learning.html>

Abstract

Koalby and Ava are two toddler-sized 3D Printed humanoid robots intended for human interaction and lab assistance. The capabilities planned for integration during this year's project include standing without support, walking while pushing a cart and lifting objects. The first task involved updating the wiring of motors and electronics to improve performance and ensure consistent functionality. Kinematics and trajectory planning were implemented to replicate human movements and simplify motion control. This was followed by the integration of sensors such as TF Luna LiDAR, Husky Lens, and IMUs to provide data required for walking trajectories and feedback control. Additional batteries and power circuits were incorporated to account for new motors and sensors. In order to test all of these aspects, a simulation model was also developed in CoppeliaSim. Finally, a user-friendly interface was created to view all the sensor data and control the humanoid robots.

Acknowledgments

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

Dr. Pradeep Radhakrishnan

Dr. Kaveh Pahlavan

Barbara Furhman

Peter Hefti

Cam Tu Lee

The 2021-2022 3D Printed Humanoid Robot MQP Team

The 2022-2023 3D Printed Humanoid Robot MQP Design Team

Table Of Contents

Abstract	2
Acknowledgments	3
Table Of Contents	4
Nomenclature	8
Abbreviations	11
Authorship Table	12
List of Figures	14
List of Tables	17
1. Introduction	18
1.1 Report Organization	19
2. 2021-2022 MQP	20
2.1 General Overview of Koalby	20
2.2 Motors	21
2.3 Electronics: Batteries and Control	23
2.3.1 Batteries	23
2.3.2 Controls	25
2.5 Software	26
2.6 Key Differences Between Koalby and Poppy	29
2.7 Capabilities	30
2.8 Testing	31
2.8.1 Arm Tests	31
2.8.2 Routine Tests	32
2.8.3 Battery Endurance Tests	32
2.9 Recommendations for Future Work	32
3. Objectives	34
3.1 Goals	34
4 Literature review	36
4.1 Humanoid Applications	36
4.2 Kinematics	38
4.2.1 Foundations	38
4.2.2 Forward Kinematics	39
4.2.3 Inverse Kinematics	41
4.3 Trajectory Planning	42
4.3.1 Polynomial Generation	42
4.3.2 Calculation of Intermediate Points	43

4.3.3 Trajectory Execution	44
4.4 Sensors	44
4.4.1 IMU	44
4.4.2 Camera	46
4.4.3 LiDAR	46
4.4.4 Ultrasonic Sensor	47
4.5 Filter Methods	47
4.5.1 Complementary Filter	48
4.5.2 Kalman Filter	49
4.6 Stabilization	51
4.7 Cart-Pushing Studies	53
4.7.1 Stabilization Overview	53
4.7.2 Stabilization System and Calculations	54
4.7.3 Cart-Pushing Results	59
4.8 UI Prototyping	60
4.9 Simulation	62
5. Methodology	64
5.1 A Term	64
5.1.1 Understanding Koalby's Initial Status	64
5.1.2 Code Refactoring	65
5.2 B Term	65
5.2.1 Forward and Inverse Kinematics	65
5.2.2 Simulation Software	65
5.2.3 Building Koalby in Simulation	66
5.2.4 Motors	66
5.2.5 Sensors	66
5.2.6 Code Architecture Evaluation	67
5.3 C Term	67
5.3.1 Trajectory Planning	67
5.3.2 Determine the Position and Motor Angles	67
5.3.3 Gather IMU Readings and Stabilized Controls	67
5.3.4 Choose a Filter	68
5.3.5 UI Prototyping	68
5.4 D Term	68
5.4.1 Testing Stability	68
5.4.2 UI Testing	69
6. Evaluating Koalby	70
6.1 Testing Electronics	70

6.2 Testing Software	73
7. Kinematics	75
8. Trajectory Planning	83
9. Electronics and Control	88
9.1 Zeroing Motors	88
9.2 Dynamixel Versus Herkulex Motors	89
9.3 Sensors	91
9.3.1 LiDAR (TF Luna)	92
9.3.2 IMU (BNO055)	94
9.3.3 AI Camera (Huskylens)	98
9.4 Electrical Integration	101
10. Simulation Software	102
10.1 Building Koalby in Simulation	103
11. Stabilization	104
11.1 Determine Position and Motor Angles	104
11.2 Gathering IMU Readings and Stabilized Controls	106
11.3 Choosing a Filter	109
11.4 Testing Stability	109
12. Koalby's Software	113
12.1 Code Refactoring	113
12.1.1 Incorporating the Simulation	113
12.1.2 Improving Usability and Readability	113
12.1.3 Testing	114
12.2 Project Architecture	114
13. UI Prototyping	119
13.1 Pre-Recorded Movements	119
13.2 Recording New Movements	120
13.3 Evaluating Prototypes	120
13.4 More Iterations	121
13.5 Final UI Design	122
14. Discussion	126
14.1 Goals	126
14.2 Assess and Repair the Physical Robot	126
14.3 Calibrate and Integrate Motors and Sensors	127
14.4 Recreate the Robot and its Movements in Simulation	128
14.5 Develop Code for Stabilization and Movement	128
14.6 Develop a User-Friendly Interface	129
14.7 Demonstration at Undergraduate Research Projects Showcase	129

15. Conclusions	131
15.1 Broader Impacts	131
15.1.1 Engineering Ethics	131
15.1.2 Societal Impact	132
15.2 Recommendations for Future Work	132
15.2.1 Autonomy	132
15.2.2 Optimization	132
15.2.3 Increasing Capabilities	133
References	134
Appendices	141
Appendix A: Humanoid Applications	141
Appendix B: Kinematics and Trajectory Planning	142
Appendix C: Troubleshooting Document	143
Appendix D: GitHub and Simulation	144
Appendix E: Electrical Diagram	145
Reflections	146

Nomenclature

Symbol/Variable	Meaning
v, v_0, v_f	velocity, initial velocity, final velocity (for PoE: linear velocity vector)
Δx	displacement
a	acceleration
t, t_0, t_f	time, initial time, final time
θ	joint rotation angle
ω	angular velocity about each x, y, z axis
S	PoE: skew vector
I	identity matrix
q, q'	vector of joint angles, vector of angular velocities
q_0, q_f	initial joint positions, final joint positions
T_i^{i+1}	transformation matrix from link i to $i + 1$
g	gravitational acceleration
z_c	height of the CoM
x''_{COM}	acceleration of the CoM
$x_{ZMP}, x_{ZMP,k}$	the ZMP's location in the x direction, the current ZMP in k
F_{cart}	force of the cart
M_c	load mass on the cart
$CP_{x,k}$	the ongoing CP

Symbol/Variable	Meaning
v, v_0, v_f	velocity, initial velocity, final velocity (for PoE: linear velocity vector)
Δx	displacement
a	acceleration
t, t_0, t_f	time, initial time, final time
θ	joint rotation angle
ω	angular velocity about each x, y, z axis
S	PoE: skew vector
I	identity matrix
s_t	time step
τ_c	torque constant
K_a	amplitude gain
μ	coefficient of friction
f_c	arm moment of inertia
J	wrist moment of inertia
γ	IMU noise
β	IMU bias
β/s	calculated angle from gyroscope
T_s	transfer function
s	seconds
P_{xyz}	positions in the x,y,and z

Symbol/Variable	Meaning
v, v_0, v_f	velocity, initial velocity, final velocity (for PoE: linear velocity vector)
Δx	displacement
a	acceleration
t, t_0, t_f	time, initial time, final time
θ	joint rotation angle
ω	angular velocity about each x, y, z axis
S	PoE: skew vector
I	identity matrix
V_{xyz}	velocities in the x,y,z
P matrix	covariance matrix
K_p	PID proportional constant
K_i	PID integral constant
K_d	PID derivative constant
$u(t)$	PID equation

Abbreviations

Abbreviation	Meaning
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
PTZ camera	Pan-Tilt-Zoom camera
DOF	Degrees of Freedom
CSV	Comma-Separated Values
DH	Denavit-Hartenberg
PoE	Product of Exponentials
ZMP	Zero-Moment Point
CoM	Center of Mass
CP	Capture Point
LIPM	Linear-Inverted Pendulum Model
RMSE	Root Mean Square Root Error
FT sensors	Force-Torque sensors
UI	User Interface
URDF	Unified Robotics Description Format
MQP	Major Qualifying Project
CS	Computer Science
HCI	Human Computer Interaction

Authorship Table

<i>Section</i>	<i>Author</i>
Abstract	Josh
Introduction	Fin
Report Organization	Erin
2021-2022 MQP	
General Overview of Koalby	Erin
Motors	Tessa
Electronics	Tessa
Software	Casey
Key Differences Between Koalby and Poppy	Tessa
Capabilities	Tessa
Testing	Erin
Recommendations for Future Work	Erin
Objectives	Fin
Literature Review	
Humanoid Applications	Tessa
Kinematics	Josh
Trajectory Planning	Josh
Sensors	Erin
Filter Methods	Erin
Stability	Erin
Cart-Pushing Studies	Tessa
UI Prototyping	Casey
Simulation	Fin
Methodology	Josh, Erin, Tessa, Fin, Casey
Evaluating Koalby	Fin, Casey

Kinematics	Josh, Erin
Trajectory Planning	Josh
Electronics and Control	Erin, Tessa, Fin
Simulation Software	Fin
Stabilization	Erin, Casey
Koalby's Software	Casey
UI Prototyping	Casey
Discussion	Erin, Tessa, Casey
Conclusions	Erin, Tessa, Casey

List of Figures

Figure 2.1: Poppy Humanoid 1.0 Reproduced as is from [9].....	20
Figure 2.2: 25 Motors in Koalby.....	22
Figure 2.3: Two Horn Dynamixel MX-28 Motor reproduced as is from [10].....	23
Figure 2.4: HerkuleX DRS-0201 Motor CAD with the Motor Adapter Prints Attached reproduced as is from [10].....	23
Figure 2.5: 7.4V Batteries in Koalby’s Legs.....	24
Figure 2.6: 11.1V Battery in Koalby’s Head.....	25
Figure 2.7: Koalby Electronics Setup reproduced as if from [10].....	25
Figure 2.8: Serial Port Usage of Koalby reproduced as is from [10].....	26
Figure 2.9: Flowchart of Primitive Manager from Last Year.....	28
Figure 2.10: Old UI.....	28
Figure 2.11: UI Flowchart from Last Year.....	29
Figure 2.12: Key Redesign Locations on Koalby.....	30
Figure 2.13: Recorded Wave Motion.....	31
Figure 4.1: Block Diagram of the Complementary Filter reproduced as is from [36].....	47
Figure 4.2: Block Diagram of the Kalman Filter as reproduced as is from [36].....	49
Figure 4.3: Bipedal humanoid robot in inverted pendulum model reproduced as is from [40]...	52
Figure 4.4: ZMP Position Example reproduced as is from [39].....	53
Figure 4.5: ZMP Example on a Humanoid Robot reproduced as is from [39].....	54
Figure 4.6: DRC-Hubo Humanoid Platform reproduced as is from [38].....	55
Figure 4.7: LIPM with Capture Point Dynamics for Cart Pushing reproduced as is from [38]...	56
Figure 4.9: System Integration reproduced as is from [38].....	57
Figure 4.10: Residual Squared Sum Over Time for Rolling Cart “C1”, and Utility Cart “C2” with and Without Load reproduced as is from [38].....	57
Figure 6.1: Koalby’s Broken Pelvis.....	69
Figure 6.2: Electrical Short.....	70
Figure 6.3: Restricted Wires.....	71
Figure 6.4: Bolt-Mounted Arduino.....	72
Figure 6.3: Koalby’s Motor IDs.....	73

Figure 7.1: The Open-Polygon Model of Koalby’s Leg.....	75
Figure 7.2: Open-Polygon Model of Koalby’s Leg.....	76
Figure 7.3: Flowchart of Forward Kinematics Code Functionality in Koalby’s Software.....	78
Figure 7.4: Flowchart of Inverse Kinematics Code Functionality in Koalby’s Software.....	79
Figure 7.5: Koalby’s Arm Straight in Zero Position in Simulation.....	79
Figure 7.6: The open-polygon model of Koalby’s arm.....	80
Figure 7.7: Flowchart of Arm MATLAB Code.....	81
Figure 8.1: Flowchart of Trajectory Planning Code Functionality in Koalby’s Software.....	82
Figure 8.2: Example of Software-Generated Points Between a Beginning and Start Trajectory in {motor id: motor position} Format to Be Fed Back to the Robot for Movement.....	83
Figure 8.3: Progression of Koalby’s Most Stable Step With Corresponding Joint Angle Values for Each Stage.....	85
Figure 9.1: Changes in Motor Values From Zeroing.....	86
Figure 9.2: Dynamixel MX-64AT Motor.....	87
Figure 9.3: HerkuleX DRS-0601 Motor.....	87
Figure 9.4: HerkuleX DRS-0101 Motor.....	88
Figure 9.5: Dynamixel AX-12 Motor.....	88
Figure 9.6: TF Luna Sensor Testing Setup.....	90
Figure 9.7: TF Luna Sensor Testing Graph.....	90
Figure 9.8: IMU Placed on CoM.....	91
Figure: 9.9: BNO055 IMU Testing Setup.....	92
Figure 9.10: Acceleration of the IMU at rest in the X direction.....	92
Figure 9.11: Acceleration of the IMU at rest in the Y direction.....	93
Figure 9.12: Acceleration of the IMU at rest in the Z direction.....	93
Figure 9.13: Acceleration of the IMU moving in the left direction slowly.....	94
Figure 9.14: Acceleration of the IMU moving in the left direction fast.....	94
Figure 9.15: IMU Coordinate system.....	95
Figure 9.16: Paper with Color Tape for Training.....	96
Figure 9.17: Huskylens trained on two colors.....	96
Figure 9.18: Correct Classification of Object.....	97
Figure 9.19: Blue Arrow Attempting to Follow Black Line.....	97

Figure 10.1: Initial Koalby Model in CoppeliaSim.....	100
Figure 11.1: Koalby’s Most Stable Position.....	102
Figure 11.2: Koalby standing in simulation with a cart.....	103
Figure 11.3: Gyroscope and Accelerometer Placed at Center of Mass.....	104
Figure 11.4: Koalby stable in simulation with PI control.....	105
Figure 11.5: Koalby Standing with cart.....	105
Figure 11.6: Flowchart of the IMU Readings with the Kalman Filter.....	106
Figure 11.7: Flowchart of the IMU Readings with the PI Controller.....	106
Figure 11.8: Koalby Standing in the Real World.....	108
Figure 11.9: Koalby Standing with a Cart and Waving.....	109
Figure 11.10: Koalby Standing Assisted by Only One Finger.....	109
Figure 12.1: Code architecture.....	112
Figure 12.2: Koalby waving from the UI.....	112
Figure 12.3: Overall code design.....	114
Figure 13.1: First UI LoFi Prototype Mockup of Pre-Recorded Movements (left) and Queue Editing Page (right).....	115
Figure 13.2: Creating Movement Page.....	116
Figure 13.3: Second Iteration UI Prototype of Described Movements.....	117
Figure 13.4: First Software UI Iteration.....	118
Figure 13.5: Final UI Homepage Design.....	119
Figure 13.6: Pre Recorded Movements Page.....	120
Figure 13.7: Final UI Record New Movement Design.....	121
Figure 14.1: Mechanical and Materials Engineering Department Presentation.....	126

List of Tables

Table 2.1: Comparing Dynamixel MX-28 and HerkuleX DRS-0201 Motors.....	22
Table 2.2: Koalby Commands.....	27
Table 4.1: Humanoid Robot Application Examples.....	36
Table 4.2: Each DH Parameter and How to Calculate Them.....	39
Table 4.3: RMSE of Experimental X-ZMP vs. Calculated X-ZMP reproduced as is [38].....	59
Table 7.1: The DH Table for the Open-Polygon Configuration of Koalby’s Leg Shown in Figure 7.1.....	75
Table 11.1: Stability Trials With and Without PI Control.....	107
Table 12.1: Initialization Time Trials.....	113

1. Introduction

The first major wave of robotics development came in the post-war industrial boom of the 1950s [1]. These simplistic machines were designed to accomplish a single, highly-specific task using a combination of early computing methods and mechanical design. They quickly revolutionized the assembly line manufacturing industry, adopted by the automotive giant, Ford [1]. As time progressed and both software and hardware engineering improved, the tasks that robots were capable of achieving grew more varied and complex, from welding to performing surgery [1,2].

Since the 1970s, the challenge of building a human-like robot has held the attention of the robotics industry [3]. From a mechanical perspective, the most unique aspect of this design is the use of bipedal locomotion [4]. This requires more robust capabilities in self-balancing and environmental response than a typical wheeled or stationary robot. As a result, humanoid robots often make extensive use of advanced sensors, analysis software, and even in recent years, learning AI [4]. These robots can serve a wide variety of functions, ranging from public relations and human-facing services to working in hazardous environments [4,5].

In the 2021-2022 academic year, a team of WPI students worked to adapt the open-source, 3D-printed Poppy robot into a more affordable version called Koalby [10]. The Poppy project was created with the objective of creating a robot that could be replicated from scratch by anyone with access to the design and the appropriate resources, i.e. a 3D Printer and commercially-available motors [6]. While the Poppy team successfully achieved their goal and had provided both open-source software and 3D hardware models, the Koalby team observed that the construction costs of the robot were quite high. Thus, they set out to redesign the robot to use less expensive options, both for printing materials and for the motors providing movement [7].

For this year, the team aimed to increase Koalby's capabilities to allow for the robot to act as an assistant in a classroom or lab setting. This would entail increasing Koalby's autonomy, including self-balancing, walking, and real-time responses to its environment. In Chapter 3, we outline a more detailed plan and decision making process that was followed throughout the year.

1.1 Report Organization

This report is organized as follows. Chapter 2 discusses the 2021-2022 MQP responsible for building Koalby and laying the foundations for the 2022-2023 team. Chapter 3 describes the objectives of the project. Chapter 4 discusses a review of literature and topics necessary to give background on the project. Chapter 5 discusses a series of methods performed in order to achieve the main project goals. Chapter 6 discusses the evaluation of Koalby from last year's MQP project. Chapter 7 discusses the inverse and forward kinematics calculated for the arm and the leg chains. Chapter 8 discusses the trajectory planning for moving the cain and the end effector in a controlled manner. Chapter 9 discusses the added electronics and sensors to the new circuit design. Chapter 10 discusses the simulation software used to implement and test code throughout the project. Chapter 11 discusses the steps taken in order to get a stabilized humanoid robot. Chapter 12 discusses the overall design and organization of the humanoid code base. Chapter 13 discusses the design and testing of the UI for Koalby. Chapter 14 is the discussion for all the results of the project testing. Chapter 15 is the conclusion to end the research paper.

2. 2021-2022 MQP

This chapter starts with understanding the current state of the project, as it is continued from last year. Last year's MQP team replicated the Poppy project, making adjustments to reduce the cost and adding an internal power source. These adjustments increased the accessibility of the robot, which could be used for educational purposes, mainly in the research and development of bipedal humanoid robots.

2.1 General Overview of Koalby

This project is the second iteration of the 3D-Humanoid Robot MQP project. Last year, an MQP team used the open-sourced Poppy Project created by the Flowers Laboratory as a guide and aimed to build a more cost-effective version of the humanoid robot. The Poppy humanoid is a 25-degrees-of-freedom (DOF) robot with a fully articulated vertebral column and comes with an embedded board that controls the motors and the sensors [8]. Poppy uses Dynamixel motors, a brand of smart actuator developed by ROBOTIS [8]. Figure 2.1 below shows the Poppy Humanoid 1.0.

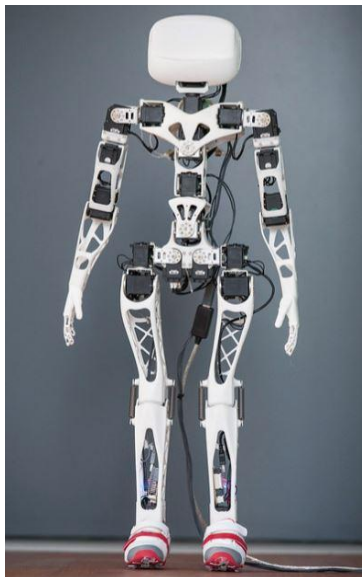


Figure 2.1: Poppy Humanoid 1.0 Reproduced as is from [9]

This project's software is based on a Python library called pypot. The Poppy libraries have been written in Python to allow for fast development and the ability to utilize other existing scientific libraries [8]. The open-source nature of this project eases accessibility to the construction and research of bipedal robots to a larger population. Poppy consists of 3D-printed parts with a modular design in order to switch out or add components to the robot easily. Poppy's design team spent the longest time on the overall design of the legs as they are made up of the hip, thigh, and feet [10]. Poppy was developed with similar proportions to a human, with a focus on replicating the hip width and gait [10].

In addition to Poppy's purpose of being an educational tool for the further development of humanoid robots, there are different applications that Poppy has been deployed in. The first one is in the School of Moon, a play where the stage is shared with children, two dancers, three NAO robots, and two Poppy humanoids. This helps bridge the gap between humans and robots. Another application of Poppy is in the Cherry Project, a community project to help reduce the feeling of isolation of children in hospitals. The Poppy humanoid aids by acting as a companion for hospitalized children in the primary school age group. This application helps mediate the barriers between the child and their friends or family as Poppy can talk to or play games with the child [8].

2.2 Motors

To reduce cost of Koalby, the Dynamixel MX-28 motors used in the Poppy project were swapped for the cheaper HerkuleX DRS-0201. As of last year's team's work, the motors used in Koalby are two Dynamixel AX-12 motors in the neck, four Dynamixel MX-64AT motors in the abdomen and hips, and 19 HerkuleX DRS-0201 motors throughout the limbs which is shown in Figure 2.2. This differs from the original Poppy project which had Dynamixel MX-28AT motors throughout the body. All 19 of the Dynamixel MX-28 motors were replaced with the HerkuleX DRS-0201 reducing the total cost by ~\$2500. The motor comparison between Dynamixel MX-28AT and HerkuleX DRS-0201 are shown in Table 2.1.

Table 2.1: Comparing Dynamixel MX-28 and HerkuleX DRS-0201 Motors

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

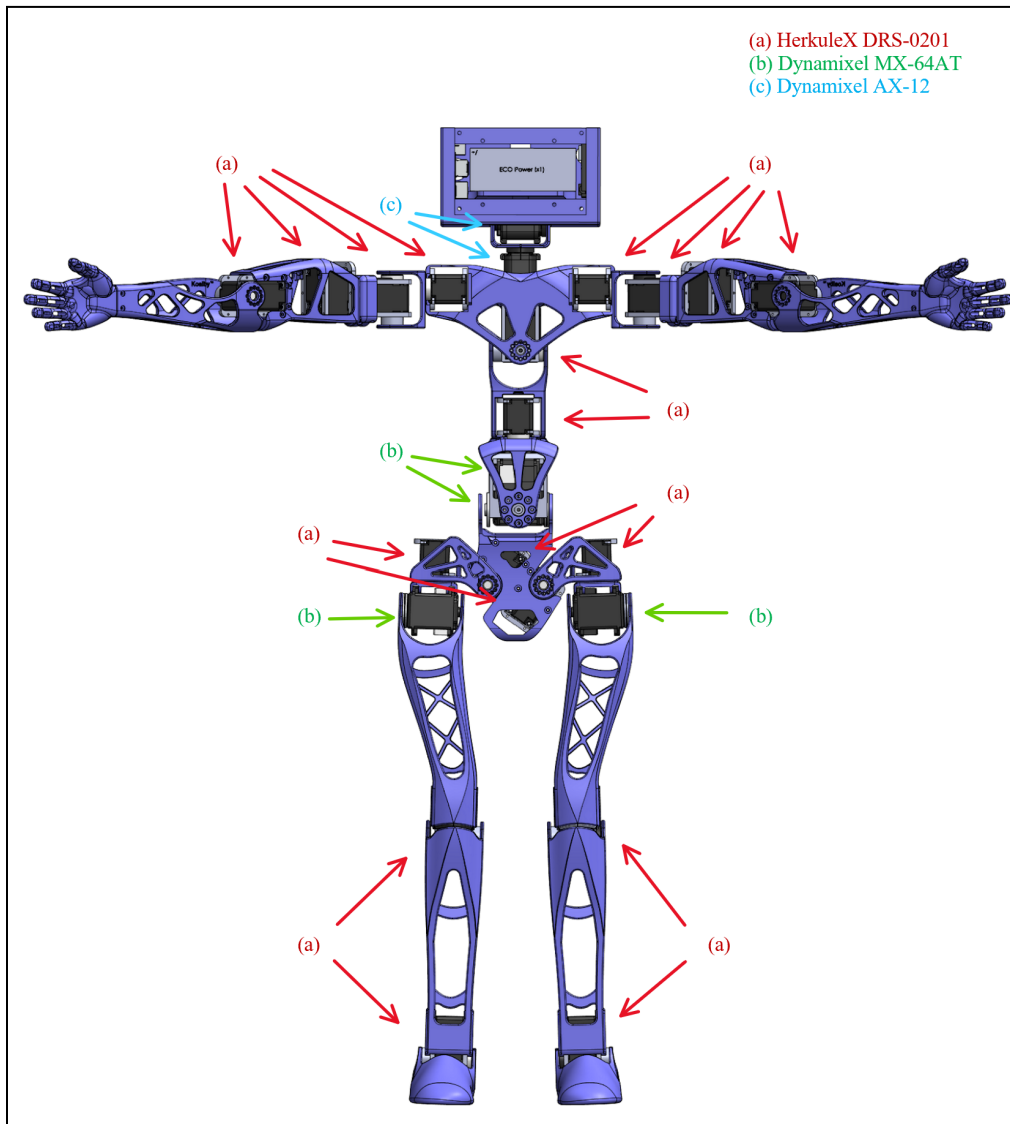


Figure 2.2: 25 Motors in Koalby

The HerkuleX DRS-0201 motors (24.0 x 45 x 31 [mm]) are significantly thinner and slightly smaller than the MX-28 motors (32 x 50 x 40 [mm]) in most other dimensions. This meant they fit in the space previously filled by MX-28 motors, but their mounting holes do not align with the existing mounting points. A motor adapter was designed and printed to realign the mounting holes, and a spacer was designed to allow the HerkuleX motors to fit in the spaces originally designed for Dynamixel motors. Figure 2.3 shows the original two horn Dynamixel MX-28, and 2.4 shows the HerkuleX DRS-0201 motors with the motor adapter prints attached.

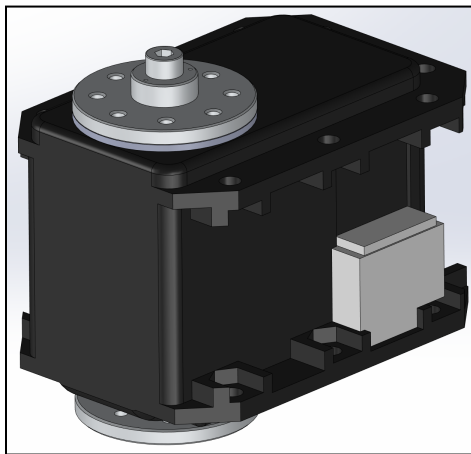


Figure 2.3: Two Horn Dynamixel MX-28 Motor reproduced as is from [10]

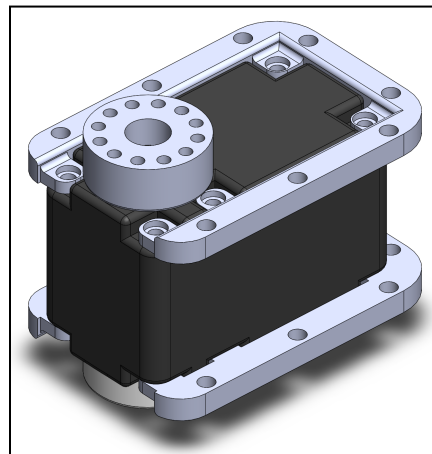


Figure 2.4: HerkuleX DRS-0201 Motor CAD with the Motor Adapter Prints Attached reproduced as is from [10]

2.3 Electronics: Batteries and Control

2.3.1 Batteries

Koalby differs from Poppy in that the power is supplied by onboard batteries rather than being plugged into a wall outlet [10]. The control wiring runs signals to the Arduino from the separate HerkuleX and Dynamixel bus systems. Koalby was powered with two 7.4V batteries and one 11.1V lithium polymer battery. The two 7.4V batteries have a capacity of 5200mAh and contain two cells. Figure 2.5 shows where the 7.4V batteries are stored in Koalby's shins. The third 11.1V battery has a capacity of 2200 mAh with three cells. Figure 2.6 shows where the

11.1V battery is stored in Koalby's head. The two 7.4V batteries were placed in parallel to power the Herklux motors (which can run on anywhere from 6-9V), a Raspberry Pi 3 and an Arduino Mega. The Arduino Mega has an integrated voltage regulator, so it can be powered directly from the 7.4V batteries. The Raspberry Pi does not have a voltage regulator, so a LM7805CV linear voltage regulator (which supplies up to 1.8A) was used to power the Raspberry Pi, which needs 1.6A to operate. The third 11.1V battery was used to power the Dynamixel motors and Dynamixel Shield.

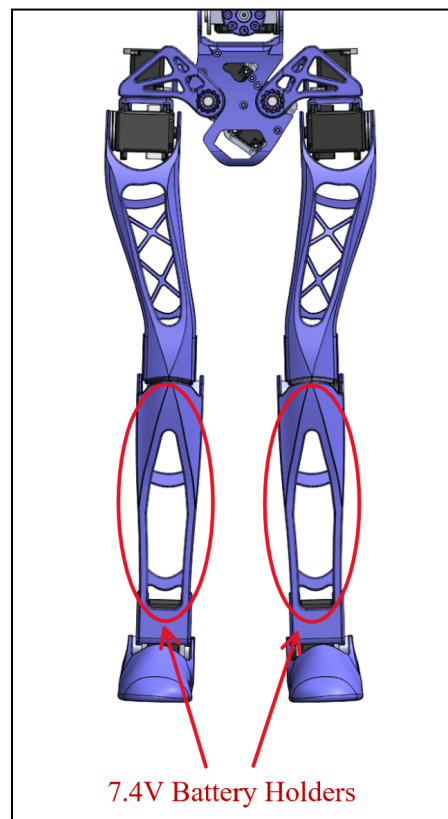


Figure 2.5: 7.4V Batteries in Koalby's Legs

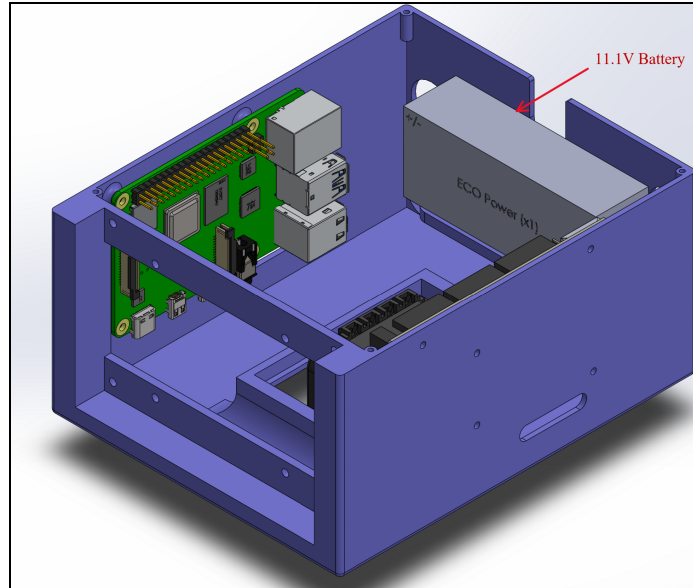


Figure 2.6: 11.1V Battery in Koalby's Head

2.3.2 Controls

The Poppy robot connected the Dynamixel motors directly to the Raspberry Pi via a custom built PCB; this custom PCB connected directly to the GPIO pins of the board [10]. This board was not available in the US and the HerkuleX motors used a different four wire bus standard than the three wire Dynamixel setup [10]. Therefore, an Arduino Mega was used in Koalby to replace the smaller adapter board. The HerkuleX motors can be controlled directly by the Arduino, while the Dynamixel motors require an additional shield, a Dynamixel Motor Shield. Figure 2.7 shows the electronics setup of Koalby, as previously described.

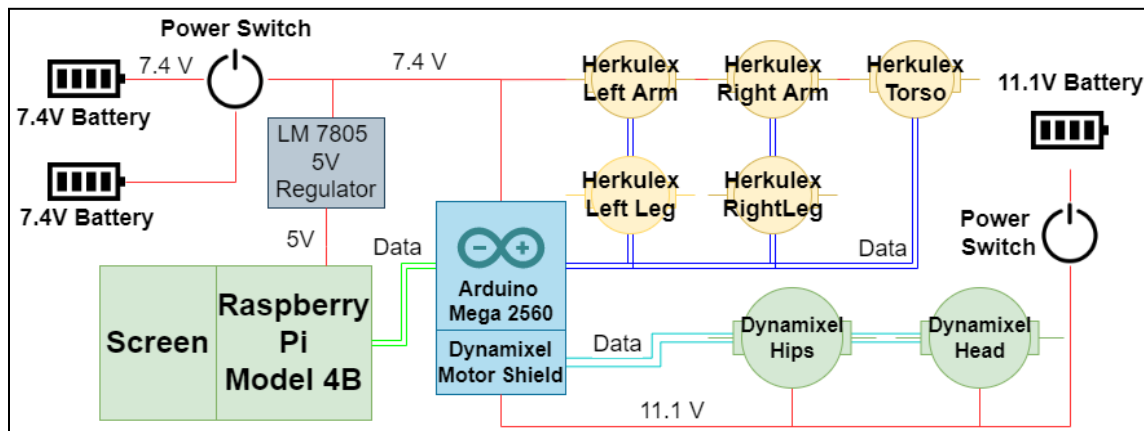


Figure 2.7: Koalby Electronics Setup reproduced as if from [10]

Serial communication between the Arduino and Raspberry Pi was accomplished via a USB-serial adapter. This was necessary because the Arduino uses the serial pins for the USB port. The adapter connected pins from the Raspberry Pi's USB port to the Serial2 pins of the Arduino. Figure 2.8 shows Koalby's serial port usage.

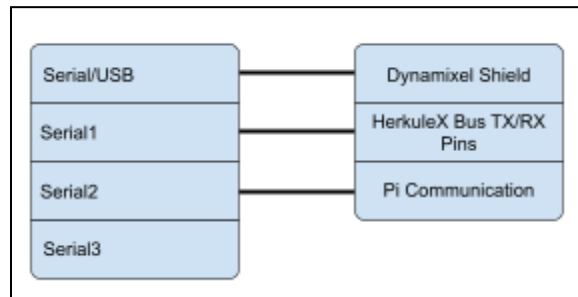


Figure 2.8: Serial Port Usage of Koalby reproduced as is from[10]

2.5 Software

The code structure of Koalby consists of Arduino code written in C++ and Python code. The Arduino code is the direct controller for the robot, moving each individual motor to its specified position. It consists of two main parts which are a setup section and a loop section. The setup section initializes various different parts of the code for use throughout its execution. The first is the serial communication port, opening the connection between the Arduino code and the Python code. Second, a robot object used to control the robot is initialized. After the setup function is complete, the loop section is continuously executed. The loop contains a switch case statement used for interpreting signals from the Python code. When a signal is sent, it is parsed and the switch case statement directs the Arduino code to a designated method within the robot class. These designated methods are used to set and get motor positions as well as sensor data.

The communication between the Arduino and the Python is simple. If the Arduino is connected to the processor running the Python code, all signals can be sent through that serial port. The Python software sends a numerical code corresponding to what it wants the Arduino to do. The Arduino then decodes the message it was sent and executes based on the switch-case statement as mentioned above. This numerical code table can be seen in Table 2.2.

Table 2.2: Koalby Commands

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 contains a list of executable functions that allow the robot to move. These functions all used a shared file called `ReplayPrimitive` that reads desired motor positions and sets the robot's motor angles. This can be seen in Figure 2.9. Once completed, the primitive manager is updated. This is where the motor positions are given to the robot class and this robot class sends information to the motor class for the motors. Then, the motor class communicates with the Arduino to send information to the actual motors.

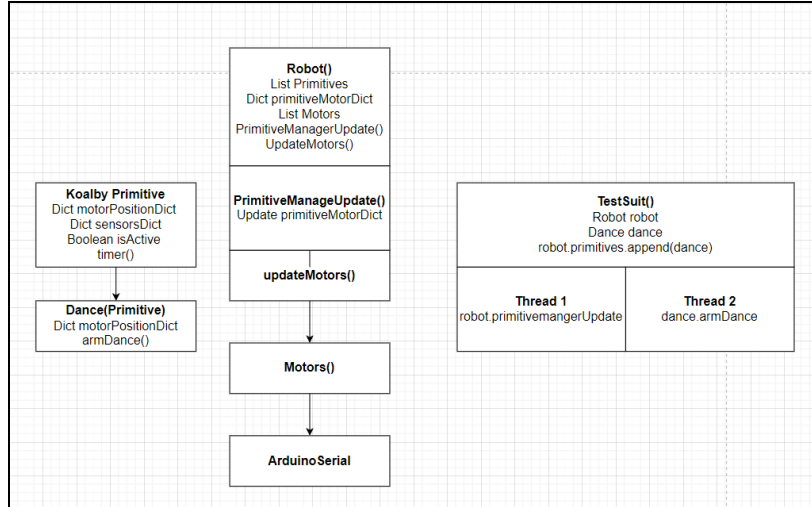


Figure 2.9: Flowchart of Primitive Manager from Last Year

Lastly, Koalby’s user interface from last year, seen in Figure 2.10, was created conceptually but did not have any functional impact on the robot’s operation. The idea was to have it threaded so the UI could be running in one thread and when a button was clicked, the robot’s movement could be executed in a separate thread. The flow chart of this can be seen in Figure 2.11.



Figure 2.10: Old UI

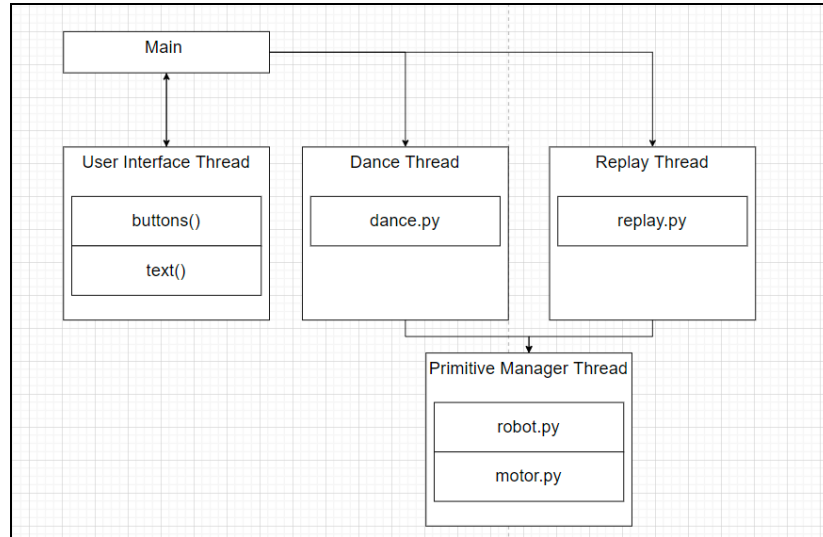


Figure 2.11: UI Flowchart from Last Year

2.6 Key Differences Between Koalby and Poppy

As explained in Chapter 2.2, several Dynamixel motors were replaced by new HerkuleX motors in Koalby. This motor change required various design modifications to fit the new motors. In addition to the motor mount that was created to fit around the HerkuleX DRS-0201 (see Chapter 2.2), a middle linking piece was created to connect the HerkuleX DRS-0201 motors to link to motors together in the abdomen to accomplish the original functionality allowing for rotation on two separate axes. Also, the servo horn patterns were redesigned from an 8-hole design to a 12-hole design to accommodate the motor change from Dynamixel to HerkuleX. Additionally, minor changes were made to the chest and shins to fit the 3D printing beds properly. Figure 2.12 shows the location of the major changes.

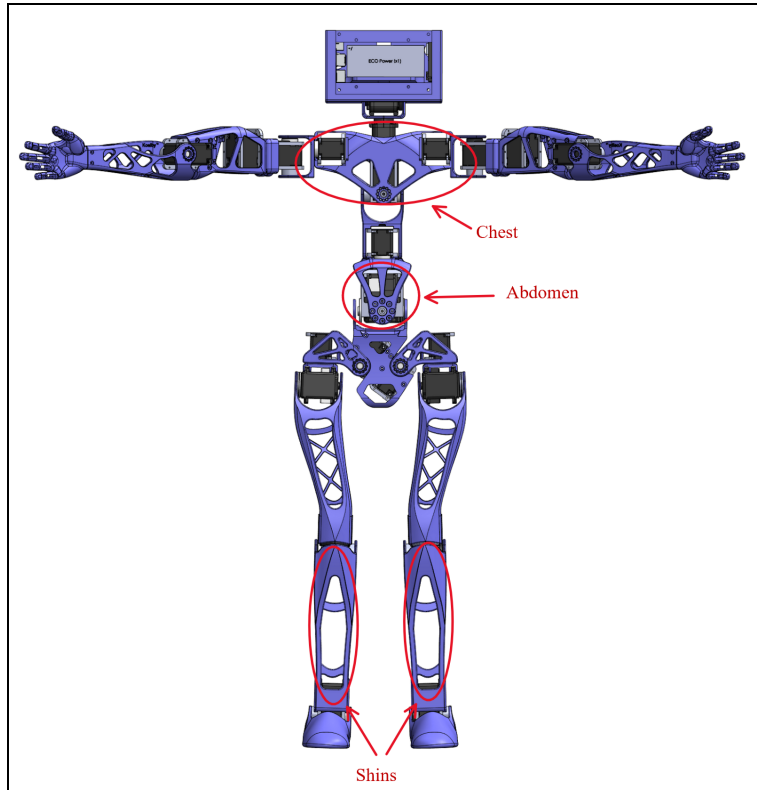


Figure 2.12: Key Redesign Locations on Koalby

2.7 Capabilities

The 2022 3D Printed Humanoid Robot MQP team successfully accomplished the goals put forth by its team. Koalby demonstrated the ability to build an open source robot. Motor changes were made to successfully decrease the cost by ~\$2,500. These motor changes were integrated into the previous design with the aid of a motor mount to fit the old style of motors. With the addition of internal batteries, Koalby can operate untethered from an external power source allowing for a wider range of functions. Primitive actions, like shaking hands and waving, were successfully recorded with the ability to be replayed. Figure 2.13 shows Koalby performing the recorded waving action. These actions were successfully demonstrated at the TouchTomorrow and WPI Project Presentation Day events.

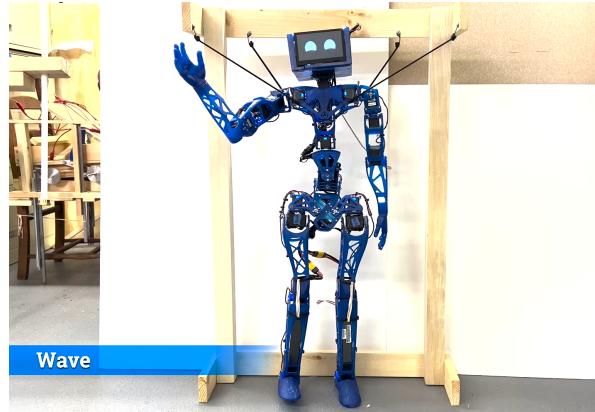


Figure 2.13: Recorded Wave Motion

2.8 Testing

Last year's team focused on three main areas of testing: arm, routine, and battery endurance testing.

2.8.1 Arm Tests

After assembling the 3D printed arm, last year's team performed motion and load tests on the right arm. The motion tests were to ensure the code was mapped to the correct motor numbers and that the motion of the arm moved to the desired positions as expected. In order to accomplish this, the team created a wave motion that the robotic arm would travel along. This aided in the team's endeavors to map motor values to desired movements[10].

The next set of testing was load bearing. This testing was performed to determine the maximum weight and stall torque the HerkuleX motors could endure before failing. This test was set up by attaching a small bottle to the hand using a rope and tested different weights by filling the bottle to different volumes[10]. Through their testing, they found that Koalby could easily lift a weight of ~175g without any issues but found a weight of 200g to create issues with torque in the motors.

2.8.2 Routine Tests

The next set of testing was not in a formal setting as mentioned in Chapter 2.8.1, these tests were done through frequent use of the robot. One of the biggest issues last year's team had was the breakage of the printed parts and the hardware mounting point. They found that unevenly distributed loads from the mounting point would cause the part to snap[10]. The way they managed to reduce breakage was by distributing the loads by having four main points to screw parts together.

In addition to redesigning parts of Koalby, last year's team found that wire management was incredibly difficult and caused a routine issue when using Koalby. They found that the wires would rub against the printed parts which would degrade the wire insulation and without the insulation they were prone to shorting which causes major damage to the electrical system of the robot. Then, in order to repair the damage caused, the wires would have to be replaced causing a delay and increasing costs. The team resolved this issue by routing the wires away from the rotating joints and by providing additional slack in the wire[10].

2.8.3 Battery Endurance Tests

The team wanted to test the endurance of the batteries and used the TouchTomorrow event, hosted by WPI, to test the durability of the batteries in the head and legs. The event ran for seven hours and the robot was running continuously as people would pass by and learn about the project. Previous endurance testing had the batteries lasting for 2 hours, however, the TouchTomorrow event was a success and Koalby was able to run continuously for the entire event[10].

2.9 Recommendations for Future Work

Last year's team was able to design and produce Koalby using the Poppy project as a starting point. However, there were areas that they were not able to get to on the software side. The first recommendation was integrating sensors, more specially IMUs, cameras, and pressure sensors. These would allow Koalby to measure its orientation within the real world and help with self-balancing. The pressure sensors would allow for precise grasping and detecting when a foot was on the ground completely [10].

Next was implementing kinematics into the code. The movements created were not based on kinematics so the robot could not see the positions in space [10]. By adding kinematics and trajectory planning Koalby would have more precise and controlled movements without needing to map predesigned movements to a specific motor angle.

The final recommendation for future work was to implement controls and algorithms for self-balancing. Koalby was able to mostly stand upright with little assistance but in order to get consistent balance and future walking, balancing algorithms coupled with sensor feedback for success [10].

3. Objectives

The overall objective for the year was originally to allow the robot to walk with the assistance of a cart and pick up and place objects. These two tasks combined would allow the robot to act as an assistant in a classroom or laboratory setting. In order to accomplish this, the team outlined a set of smaller goals that, if achieved, would allow for assisted walking. To start, we wanted to familiarize ourselves with Koalby and understand what needed to be repaired in order to get the robot in working condition. Once this was completed, we wanted to introduce new motors and sensors into Koalby's hardware to increase its capabilities. A crucial element that our team wanted to focus on for this project was simulating Koalby in order to test and develop different functionalities more safely and efficiently than testing only on hardware. With regards to these functionalities, two high-level tasks we made for ourselves were to develop methods to make Koalby stand unassisted in a stabilized position, walk while assisted with a cart, and to improve the efficiency of the robot's different limb movements. Finally, once all of this was in place, the creation of a user interface (UI) would allow for the easy control of these different motions through an outside user's input.

3.1 Goals

Our first goal was assessing and improving the physical robot. The necessity of this task was immediately apparent, as in the first weeks of the project the team struggled to have the robot activate and move consistently. As adding and improving the robot's functions would not be feasible if it did not work on the most basic levels, general repair became first priority. We determined that the metric for this goal's success would be if the robot successfully turned on and entered its initialization position every time we attempted to make it do so.

Next, we set out to incorporate more sensors into Koalby's functionality. This was determined to be an essential portion of making the robot into a functional lab assistant, as it would need to be capable of accurately observing and responding to its environment. Sensors are a necessity for allowing any system to operate without constant input from a human controller. Additionally, we aimed to upgrade Koalby's motors. The robot's original design made use of two brands of motors, and some motors proved incapable of obtaining the torque necessary to move the limbs they were attached to. This posed another major obstacle to creating an autonomous

walking robot, as the increased software complexity associated with controlling different motor types reduced the efficiency of the robot's operation, and failing motors contributed to the inconsistency that the first goal aimed to solve. We set this goal's metric for success as successfully identifying, acquiring, and controlling appropriate motors and sensors that would allow for Koalby's increased autonomy.

While the first two goals were in motion, we also aimed to recreate the robot in simulation. This would allow us to test and develop new features while the physical problems still existed, as well as increasing ease of testing in the future. The remote and infinitely repeatable nature of simulation tests make them ideal for testing a robot that proved to be unreliable in the real world. Achievement of this goal would mean creating an accurate simulation of the robot, and transferring the success of a simulated feature test into an equally successful implementation of that feature in the real robot.

In simulation, and therefore in reality, the primary goal was to get the robot to stand stably. This acted as a goal that was both reasonable to achieve and acted as an indicator of success in the previous goals. The integration of sensors would be essential for allowing the robot to balance and remain standing in both reality and simulation. A high-quality and accurate simulation would be necessary to develop a standing test that could translate to reality, and the physical repairs and motor upgrades would be essential for the physical robot's success in this test. As we knew that the presentation of our project would take 15 minutes, we determined that the robot standing independently in both simulation and reality for at least 15 minutes would constitute success.

Finally, we wished to develop an interface that would allow users outside of the development team to meaningfully interact with Koalby. This is essential for the robot's functionality as a lab assistant, as it must be intuitive enough to be useful to professors. Additionally, as both Koalby and its predecessor, Poppy additionally served as educational and outreach opportunities for robotics, allowing audiences to interact with the robot would greatly increase the appeal of these sorts of demonstrations. We qualified success in this goal as being able to activate the robot and execute movements entirely through the user interface.

4 Literature review

Then, the literature review was completed to determine the goals and applications of this project based on previous humanoid robots. 4.1 describes humanoid applications in industry, 4.2 describes the kinematics, mainly inverse and forward, 4.3 describes trajectory planning, 4.4 describes the sensors, 4.5 describes filters, 4.6 describes stabilization, 4.7 describes cart-pushing studies, 4.8 describes UI prototyping, and 4.9 describes simulation.

4.1 Humanoid Applications

In order to have a clear objective for this project, current humanoid robot applications needed to be explored. Some of these applications include medicine, industry, service, space exploration, and outreach. Various humanoid robots were examined to understand possible applications within these industries (Table 4.1). Photos of each of the Humanoid Robots are in Appendix A.

Table 4.1: Humanoid Robot Application Examples

Industrial Field	Service / Medical	Space	Outreach/Interaction
Digit (Ford Agility Robotics)	T-HR3 (Toyota) - mobility service	Vyammitra	Sophia (Hanson Robotics) - human/robot interaction
Nextage (Kawada Robotics)	Kime (Macco Robotics) - bartender	Fedar	Surena Robot (Iranian U) - inspire students
	Robothespian - actor	Robonaut 2 (NASA)	
	Smart Field Hospital	Valkyrie (NASA)	

In the industrial field, humanoid robots have been used to assist in warehouse management and maintain production for manufacturing companies. For example, Digit, created by Agility Robotics, was incorporated into a factory setting by Ford [12]. Digit is a headless humanoid robot that can navigate stairs, obstacles, varied terrains, balance on one foot, pick up and stack boxes weighing up to 40 pounds, and fold itself for compact storage. The future application envisioned for Digit is to assist in package deliveries; Digit would ride in a driverless car and deliver packages to customers, automating the entire delivery process. A second example of humanoid robots used in industry is Nextage by Kawada Robotics [13]. Nextage was developed to perform maintenance tasks alongside human workers in industrial settings. This robot was designed as only a torso with two 6 DOF arms for high functionality in process management and object manipulation [11].

Humanoid robots have also been designed for acts of service ranging from medical aid to bartending and entertainment. In the medical field, humanoid robots have been used at the Smart Field Hospital in Wuhan, China. This usage started in March 2020 during the COVID-19 pandemic [11]. During such times, humanoid robots could relieve overworked nurses to do basic cleaning and delivery tasks. These robots are also being used as medical assistants to disinfect surfaces, measure temperatures, deliver food and medicine, and entertain medical staff and patients. Additionally, the T-HR3 by Toyota was designed to provide service and skills, such as surgery, while operated by a person located elsewhere. This humanoid robot can mimic the movements of its human operators and walk. [11].

Kime by Macco Robotics was designed to be a food and beverage serving robot with a human-like head, torso, and arms. Kime was tested at gas stations throughout Europe and in a Spanish brewery; this robot can serve up to 300 glasses per hour and has 14-20 degrees of freedom, smart sensors, and uses machine learning to improve its skills [14]. For entertainment, Robothespian is a robot actor that comes with a library of impressions, greetings, songs, and gestures [11]. Multiple Robothespians can be incorporated to become a robotics theater with movement tracks, animation software, touchscreen control, lighting, and sound.

Furthermore, several humanoid robots have been developed for space exploration research, termed “robonauts”. Two key examples developed by NASA include Robonaut 2 and Valkyrie. Robonaut 2 successfully traveled to space and spent seven years on the International Space Station [11]. Valkyrie is a more recent robonaut designed to withstand harsh environments

similar to those on the moon and Mars [11]. Developed by the Indian Space Research Organization, Vyommitra, another humanoid robot, was intended to conduct microgravity experiments to help prepare future crewed missions [11]. Lastly, Fedar by Final Experimental Demonstration Object Research was a Russian remote-controlled humanoid that flew to the International Space Station in 2019 [11]. Fedar simulated repairs during a spacewalk and later returned to Earth.

Humanoid robots have also been developed for research and collaborative purposes. Sophia, by Hanson Robotics, is a social humanoid robot who serves as a robotic ambassador to advance research related to robotics and human-robot interactions [15]. Sophia can move, talk, show some emotions, draw, and sing. Additionally, Surena Robot by Iranian University of Tehran is an adult-sized humanoid robot capable of face and object detection, speech recognition and generation, and can walk with a speed of 0.7 km/hr [16]. Surena has 43 DOF and hands that can grip different shapes. It is currently being used to research bipedal locomotion, artificial intelligence, and for outreach to attract students to careers in engineering.

Overall, this research shows the various humanoid robot applications that are currently being explored including industry, service, and outreach. Based upon this, our team decided to focus our application towards service as a lab assistant. This is because service applications have positive broader and societal impacts and humanoid robots in lab settings are currently less explored compared to factory and medical settings.

4.2 Kinematics

4.2.1 Foundations

The implementation of kinematic methods is paramount to moving a robot's numerous limbs. The correct use of kinematics allows the user to determine the specific motions that lead to a certain position of a robotic body part. Implementing kinematic functionality gives the angle values of the joints that lead to certain positions of the end effector of an arm or leg (known as a kinematic chain with each section of the limb between joints referred to as a link), and vice-versa [17]. In fundamental physics courses, students are taught kinematics and learn the four kinematic equations (1) through (4). These equations are the foundations of the kinematic equations that will be used in achieving bipedal locomotion.

$$v = v_0 + at \quad (1)$$

$$\Delta x = \left(\frac{v + v_0}{2}\right)t \quad (2)$$

$$\Delta x = v_0 t + \frac{1}{2}at^2 \quad (3)$$

$$v^2 = v_0^2 + 2a\Delta x \quad (4)$$

4.2.2 Forward Kinematics

Forward kinematics takes the angle positions of each joint (the joint space), and converts those values to Cartesian coordinates representing the end-effector's position (the task space). Inverse kinematics is responsible for the opposite, as it takes values in the task space and converts them to the corresponding joint space values [18]. Specifically for humanoid robots, the use of inverse kinematics is especially important for its UI and ease of use, as a limb can be told to move to a certain desired Cartesian location, and this location can be converted into the motor positions needed to get to that location.

Forward kinematics is the less complicated of the two to calculate. To start, a transformation matrix is found for each link that describes movement from one position to another. The general form of this matrix can be seen in equation (5), and its calculation is done by finding the Denavit-Hartenberg (DH) parameters of each link within the chain [17]. Each DH parameter and how to find each are shown in Table 4.2.

$$T_0^1 = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \cdot \cos(\alpha_i) & \sin(\theta_i) \cdot \sin(\alpha_i) & a_i \cdot \cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \cdot \cos(\alpha_i) & -\cos(\theta_i) \cdot \sin(\alpha_i) & a_i \cdot \sin(\theta_i) \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Table 4.2: Each DH Parameter and How to Calculate Them.

Parameter	Meaning/How to Calculate
d	distance along the z-axis from the chosen link to the next
a	distance along the x-axis from the chosen link to the next
θ	the angle to rotate around the chosen link's z-axis
α	the angle to rotate around the chosen link's x-axis

This process of calculating a transformation matrix is repeated for every link in the kinematic chain [17]. Since the end goal of a forward kinematics problem is to determine the location of the end effector, a transformation matrix that describes the motion of the end-effector in relation to the base is found by iteratively multiplying the transformation matrices of each link together from the base joint to the end-effector [18].

Another way to solve for the forward kinematics is through the product of exponentials method (PoE). This method is more complex than the DH method but is more consistent with the answer obtained, as with DH parameters there can be many different answers for the same manipulator. Marking the different frames of the robot for PoE is very similar to the DH method, but it focuses more on angular and linear velocities rather than different angle and displacement parameters [19].

This process begins with finding the matrix that represents the home configuration chosen for the manipulator. This is done by comparing the end effector frame with the base frame and constructing a 4x4 transformation matrix representing any rotation or translation.

Then, twist vectors need to be calculated for each joint in the manipulator. These 6x1 vectors are made up of two 3x1 vectors, w and v . Vector w is the vector that relates the angular velocities of a joint to the base frame, and v is a vector that does the same with linear velocities [19]. Once these twist vectors are found, the intermediate transformation matrices for each link of the manipulator are found using the equations (6) and (7) [19].

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

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

Finally, the matrix representing the transformation from the base frame to the end effector frame is found similarly to the DH method as each intermediate transformation matrix is multiplied iteratively from the base frame to the end effector frame [19].

4.2.3 Inverse Kinematics

Finding the inverse kinematics of a kinematic chain is more difficult, as there is no simple method such as calculating DH parameters to solve for joint space values from task space values. There are many ways to solve inverse kinematics, but there are two that are most effective for humanoid robots' arms and legs that are commonly used which involve the least intensive hand calculations and, in the event that analysis cannot be done by hand, are easy to input into a software such as MATLAB to solve. The first is through a geometric approach, where the kinematic chain is modeled and the different angles are solved using geometry. The second way is through an algebraic approach where different systems of equations are used to solve for the angle values. It is difficult to determine which method is better for any given application as it is very dependent on the complexity of the problem, but it is generally easier to attempt to draw the kinematic chain and execute a geometric approach if possible, as algebraic solutions can be more complex and challenging to understand [18]. Examples of full calculations using each method on a spherical manipulator can be seen in Appendix B for comparison. The equations gathered from inverse kinematics calculations as well as the transformation matrices for forward kinematics can be hardcoded in the robot's software to avoid doing these calculations every time a limb needs to be moved, so only a single initial calculation is necessary.

4.3 Trajectory Planning

For smoother movement of limbs and control of its walking motion, humanoid robots need trajectory planning implemented into their software. The process of trajectory planning and generation primarily involves taking two points, one at the beginning of the desired trajectory and one at the end, and generating a series of intermediate points along a desired path that the robotic limb will move through during its journey. Trajectory planning consists of three major steps: generating the desired trajectory polynomial, using this polynomial to obtain the intermediate points along the trajectory, then finally making the robot move to these points in sequence [20].

4.3.1 Polynomial Generation

Firstly, there are three types of trajectory polynomials: linear, cubic, and quintic. Each of these polynomials are defined based on different movement constraints, these being position, velocity, and acceleration. Linear polynomials enable control over position only, and should never be used as they can make movement unpredictable due to the absence of speed control. Cubic polynomials expand functionality by allowing control of velocity and are most widely used due to their simplicity and functionality in controlling how fast a robotic limb moves from one point to another. The final type, quintic polynomials, further allow control of acceleration but are only used for more complex functions due to the added calculations needed and are generally not needed for the majority of applications [20].

Cubic polynomials are the most useful for operating a humanoid's different limbs as they avoid the complexity of quintic functions and are effective in applications where acceleration is constant. The general form of a cubic polynomial is seen below, where t is equal to time, each a is a coefficient and $q(t)$ is the angle position of a joint as a function of time [20].

$$q(t) = a_0 + a_1 * t + a_2 * t^2 + a_3 * t^3 \quad (14)$$

To successfully generate the trajectory and calculate the coefficients of each term, the initial and final velocities and times of the trajectory must be known and substituted into equations (15) through (18). These equations represent the initial and final velocities and

positions of the trajectory, where the velocity equations are obtained by taking the derivative of the constraint equations for position [20].

$$q(t_0) = q_0 = a_0 + a_1 * t_0 + a_2 * t_0^2 + a_3 * t_0^3 \quad (15)$$

$$q'(t_0) = v_0 = a_1 + 2 * a_2 * t_0 + 3 * a_3 * t_0^2 \quad (16)$$

$$q(t_f) = q_f = a_0 + a_1 * t_f + a_2 * t_f^2 + a_3 * t_f^3 \quad (17)$$

$$q'(t_f) = v_f = a_1 + 2 * a_2 * t_f + 3 * a_3 * t_f^2 \quad (18)$$

To solve these equations for the coefficients, they can be put into matrix form as seen in equation (19), where the coefficients can be found by taking the inverse of the larger matrix and finding the dot product of that and the vector of initial and final conditions [20].

$$\begin{bmatrix} 1 & t_o & t_o^2 & t_o^3 \\ 0 & 1 & 2t_o & 3t_o^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ q_f \\ v_f \end{bmatrix} \quad (19)$$

4.3.2 Calculation of Intermediate Points

Once the trajectory polynomial is found, the second step is to generate the intermediate points along the desired trajectory. As the initial and final times and velocities are known, all that is needed is to substitute intermediate times between the desired initial time and the desired final time. The user defines how often an intermediate position will be calculated, but generally, a smaller time step is better as having more positions along a trajectory means that the movement will be more predictable. Despite this, the time step should not be too small either as it could lead to inconsistent and jittery movements as the robot goes from one position to the next [20]. The optimal time step depends on the use desired, and can only be found through testing the system.

4.3.3 Trajectory Execution

Once this is done, the final step is to have the robotic limb iterate through the generated trajectory points at the correct velocity. Movement is easily done by feeding the positions into the robot, but this does not take into account the desired velocity. To account for this, the robot can be given a delay time between moving through each position that is equal to the final time for the entire trajectory divided by the number of points on the trajectory. With this, the robotic limb will move along the desired path at the velocity/time specified by the user.

4.4 Sensors

For autonomy and increased stabilization, it is required to equip humanoid robots with a series of sensors. These sensors will allow for the collection and analysis of quantitative data to improve functionality such as walking, pushing a cart, and picking up objects. The main sensors that are implemented into autonomous robotics [21, 25, 31, 35] fall under four main categories: IMU, Camera, (LiDAR), and Ultrasonic.

4.4.1 IMU

An IMU is an electronic device that typically combines three sensors: an accelerometer, gyroscope, and magnetometer. These three sensors relate to the dynamics and orientation of the entire IMU system [22]. There are two types of IMUs, type I and type II. Type I consists of just the accelerometer and gyroscope readings where type II includes the additional magnetometer readings [21]. Type II IMUs will have 9 DOF while type I IMUs have 6 DOF. The accelerometer measures the acceleration of the system. In order to determine which direction is downward, the accelerometer also measures gravity. Furthermore, the gyroscope measures the angular rate of rotation. Lastly, the magnetometer measures the magnetic field, similar to a compass [22].

The main purpose of using an IMU is to establish a coordinate system of the robot within the world. To properly assess both translational and rotational changes in an IMU system, it is necessary to establish a coordinate system with three axes: x, y, and z. In order to achieve this, there are two coordinate frames that need to be taken into account: a local frame and the global frame. The local frame is attached to the robot and the global frame is the robot's pose with

respect to the real world [22]. The local acceleration is equal to the acceleration of the local system plus gravity in the global system. This equation can be seen below:

$$\alpha = a + g \quad (20)$$

True accelerations will typically mask gravity. They are prone to noise and bias, which is accounted for by adding them to the accelerometer equation. Bias can be accounted for, but noise is disruptive to the sensor readings and needs to be calibrated out [22]. These additions can be seen in the equation below:

$$\alpha = a + g + \gamma + \beta \quad (21)$$

A common source of error prevalent in IMUs is the noise produced by the gyroscope. There are three common attributes of the gyroscope that cause this error: inherent noise, response to linear vibration, and misalignment errors [23]. The inherent sensor noise represents the random variation seen in the gyroscope's output during operation in static inertial and environmental conditions [23]. Gyroscopes are used to measure the angular rate of rotation and their response to linear motion introduces errors in their measurements [23]. In a perfect world or in a simulation, each of the gyroscope's axis of rotation will be perfectly aligned 90° away from one another. However, this is rarely the case in the real world and thus produces another source of error [23]. These three sources of error contribute to sensor noise for the IMU and these errors can introduce drift into the IMU readings.

When looking at IMU data there is an offset, typically a small value, on the average signal output [24]. Even when there is no movement from the device, the offset can be seen. This offset is defined as the sensor bias. Additionally, due to the physical properties of the sensors in an IMU, more specifically of the accelerometer and gyroscope, the sensor readings change over time depending on the length of usage [24]. This can result in the sensor bias increasing.

Other prevalent issues in an IMU stem from the two sensors that are providing the same orientation readings. These are the accelerometer and the gyroscope, and they are usually not in agreement. The accelerometer will provide readings on the orientation angle while the gyroscope will provide readings on the change in orientation [22]. In order to overcome this problem,

sensor fusion needs to be implemented [22]. Sensor fusion is the process of taking conflicting information from different sensors and making a reasonable estimate of the truth [22].

4.4.2 Camera

Cameras are an essential type of sensor in robotics that humanoids utilize to gather visual information about their surroundings, detect and identify objects, and navigate their environment. By using the data obtained from the camera, robots can perceive their surroundings visually, recognize faces, detect motion, and navigate through their surroundings [25]. Additionally, cameras can be used to monitor and control the robot's movements and provide feedback to the operator. The types of cameras that are typically used in industry are PTZ cameras and stereo cameras [26]. A PTZ camera is a regular vision camera that can be directed and zoomed towards specific areas of interest using a structure attached to it. These types of cameras are useful as they are able to move independently of the mount and allows the robot to look in different directions. Stereo vision allows the robot to perceive depth, shape, and size of a robot by using two cameras with a known distance between them, similar to human eyes [26].

However, there are some issues associated with the use of cameras in robotics. One major challenge is the variability in lighting conditions, which can affect the camera's ability to capture clear and accurate images [27]. Another challenge is the need for advanced algorithms to process the vast amount of visual data generated by the camera.

4.4.3 LiDAR

A light detection and distance range sensor, more commonly known as LiDAR, is an active remote sensing system. This device makes use of a continuous laser [28]. To measure distance, A LiDAR device will emit infrared light pulses. Then, when the pulses reflect off of surfaces and return to the LiDAR, the distance is recorded [28]. There are two main types of sensors, a 2D and a 3D LiDAR [29]. The 2D sensor can measure the distance by bouncing off one light whereas the 3D sensor emits multiple beams at once to gather a 3D image of the object [29]. A significant advantage of using LiDAR sensors is that they are able to capture precise distance measurements and are able to capture the object's position and shape. This sensor also offers reliable detection ranging from short to long distances with high accuracy [30].

Some disadvantages of using LiDAR are that the sensor can have limitations depending on the lighting and if the environment is too dark then it will affect the level of refraction, therefore affecting the amount of pulses. On the other hand, if the environment is too bright then the pulses will not work, again due to the LiDAR working off reflection [31].

4.4.4 Ultrasonic Sensor

An ultrasonic sensor is an electronic device that detects the distance of an object through the pinging of ultrasonic sound waves between its emitter and the object. This is done by converting the waves into an electrical signal [32]. The typical ultrasonic wave signal travels at a frequency above 18kHz [33]. There are two main types of Ultrasonic sensors: proximity detection and ranging movement [34]. Proximity detection identifies an object passing within range and generates an output signal. By measuring the distance between an object moving to and from the sensor, the ranging measurement allows continuous distance measurements.[34]. A main advantage of ultrasonic sensors is that they detect objects irrespective of their surface texture or color [35]. Through this device, a robot is capable of understanding its surroundings.

Ultrasonic sensors have a high level of intensity but come with disadvantages. These types of sensors have a tendency to detect false signals disturbed by the environment. Some examples include wind from a ceiling fan, echoes, noise, and object surface textures [33]. In addition, they can only detect objects within their range and line of sight and multiple ultrasonic sensors would be needed to overcome this hurdle. Therefore this sensor is only effective in specific environments.

4.5 Filter Methods

As mentioned previously in the chapter, there are issues with noise and bias that can occur when using an IMU. A way to combat this bias and noise is to calibrate the device and apply a filter. There are two main types of filters that are used for IMUs, the complementary filter and the Kalman filter.

4.5.1 Complementary Filter

The complementary filter fuses a high pass filter and a low pass filter to eliminate the noise in the IMU [36]. During stabilization and assisted walking, there are multiple forces working on the robot. These forces are measured by the accelerometer and the small forces create disturbances in the measurements [36]. The long-term measurement is reliable, so it needs a low pass filter for correction. For the gyroscopic sensor, the integration is performed over a period of time causing the value to begin to drift in the long-term, thus a high pass filter is needed for this correction [36]. Figure 4.1 represents the complementary filter process:

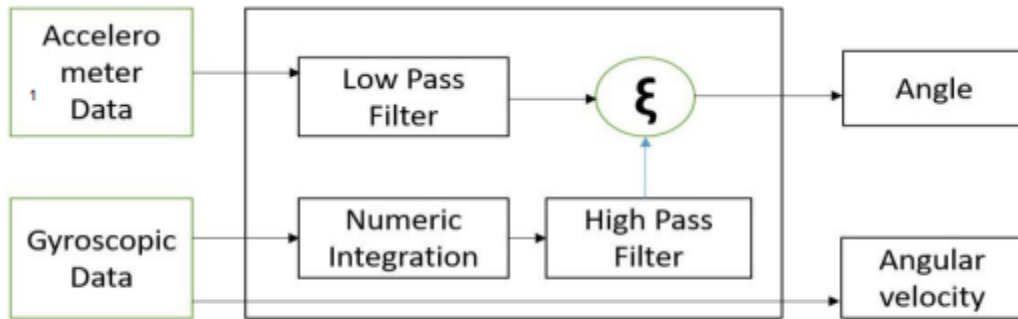


Figure 4.1: Block Diagram of the Complementary Filter reproduced as is from[36]

In the complementary filter, there are two transfer functions for each input passing through the filter:

$$\text{Low pass transfer function} = \frac{1}{1+Ts} \quad (22)$$

$$\text{High pass transfer function} = \frac{Ts}{1+Ts} \quad (23)$$

The total transfer function for the complementary filter can be seen in the equation below with a total gain of 1:

$$\text{Total} = \frac{\alpha}{1+Ts} + \left[\frac{Ts}{s(1+Ts)} + \frac{1}{s} \right] \times \beta \quad (24)$$

In the above equation, α is the acceleration data input and β/s is the calculated angle from the gyroscope after being integrated. The complementary filter is the fusion of the high pass and the low pass filters to reduce the noise seen in the IMU.

4.5.2 Kalman Filter

On the other hand, the Kalman filter is an iterative process that makes it useful in dynamic systems. This process predicts a future value by changing the previous data collected in the system [36]. The Kalman filter uses a correlation between the prediction value and the actual value to generate a prediction error, which feeds back into the system for correction on the next iteration. An advantage to this filter is that there is little memory used and it executes with little computation time [36]. Below in Figure 2 is a diagram representing the Kalman filter process:

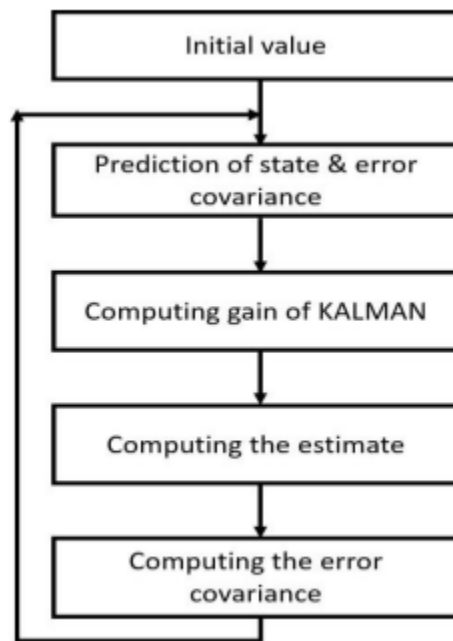


Figure 4.2: Block Diagram of the Kalman Filter as reproduced as is from [36]

There are three major components to the Kalman filter: predict, measure, and update. These are then iteratively processed while using the IMU. Within the Prediction component, there are two equations being calculated: the state vector and the covariance matrix. The state vector is used to determine where the new position is in vector form [36]:

$$X = [P_x \quad P_y \quad P_z \quad V_x \quad V_y \quad V_z] \quad (25)$$

In the below equation, vectors A and B are kinematic equations for the position, velocity, and acceleration.

$$X = Ax + Bu \quad (26)$$

The next step is determining the covariance matrix to understand the uncertainty of the new state (P). In the equation below, Q is the process noise covariance matrix.

$$P = AP * AT * Q \quad (27)$$

P should result in the following matrix:

$$P = \begin{vmatrix} \text{variance_px}, 0, 0, 0, 0, 0 \\ 0, \text{variance_py}, 0, 0, 0, 0 \\ 0, 0, \text{variance_pz}, 0, 0, 0 \\ 0, 0, 0, \text{variance_vx}, 0, 0 \\ 0, 0, 0, 0, \text{variance_vy}, 0 \\ 0, 0, 0, 0, 0, \text{variance_vz} \end{vmatrix} \quad (28)$$

For the measurement step, there are two components being calculated: the covariance matrix R and the Kalman gain. The covariance matrix is converting the sensor readings that are in the form of Z=[px,py,pz] so that they are able to be compared to the predicted value in the previous step. The R matrix that is produced can be seen below:

$$R = \begin{vmatrix} \text{variance_in_pos_x_reading}, 0, 0 \\ 0, \text{variance_in_pos_y_reading}, 0 \\ 0, 0, \text{variance_in_pos_z_reading} \end{vmatrix} \quad (29)$$

The next step is calculating the Kalman gain. The Kalman gain is used to compare the uncertainty in the prediction and measurement steps. This is done by calculating the uncertainty percentage in the predicted value to the overall uncertainty. The equation can be seen below:

$$K = (P \times HT) / ((H \times P \times HT) + R) \quad (30)$$

The H matrix in the above equation is used to convert P to the same size matrix as R. The H matrix can be seen below:

$$H = \begin{vmatrix} 1, & 0, & 0, & 0, & 0, & 0 \\ 0, & 1, & 0, & 0, & 0, & 0 \\ 0, & 0, & 1, & 0, & 0, & 0 \end{vmatrix} \quad (31)$$

In the final component of the Kalman filter, there is one equation for the final state vector. This is to determine where the actual position is. This equation can be seen below:

$$X = x + Ky \quad (32)$$

In order to get the actual value of X, the predicted value is collected and added to the difference between the predicted and measured values with the Kalman gain. The measured value should be close to the predicted value to have accounted for the correct amount of uncertainty [36]. After this step, the covariance matrix P is updated and then the process starts again.

4.6 Stabilization

Getting a bipedal humanoid robot stable is a difficult and complex task to achieve, and research is being conducted on the best approaches to this challenge. Some institutions that are currently doing research include the Georgia Institute of Technology, the University of Osaka, the University of Tokyo, and the University of Sherbrooke [37].

There are many aspects that go into stabilization, which is to keep the centerline of mass at the center of bearing mass [38]. This prevents the robot from falling to one side or the other. One major aspect of stabilization, especially when researching bipedal locomotion, is looking at the gait cycle. There are two phases in one gait cycle: stance and swing [39]. Stance occurs when the entire foot is on the ground, and swing is when the foot and knee are in the air. During regular walking motion, movement is usually 60% stance and 40% swing [39]. Throughout this motion, the center of mass, which is located about one-third of the distance between the hip joint and the shoulder, must stay at the center of bearing mass [39].

Zero-moment point control is a method, along with trajectory tracking and inverse kinematics, that is used in the implementation of stable walking in bipedal robots. When on a flat surface, the Zero-Moment Point (ZMP) is the same as the center of pressure [39]. The ZMP is the point on the ground where the torque is produced as a result of the inertial and gravitational

forces. The current standard for bipedal gaits is based on the linear inverted pendulum where the foot on the ground acts as the fulcrum, the leg attached to the foot on the ground as the rod, and the upper body as the mass [10]. Figure 4.3 shows a diagram showing the inverted pendulum model:

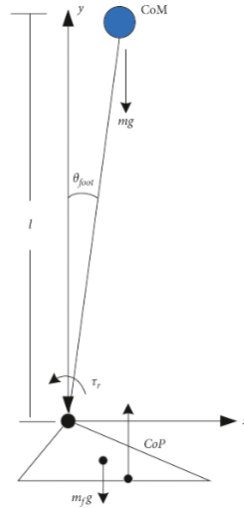


Figure 4.3: Bipedal humanoid robot in inverted pendulum model reproduced as is from [40]

During bipedal locomotion, a controller is required to stabilize the robot while in motion and while stationary. Using an IMU, a proportional, integral, and derivative (PID) controller can be implemented to stabilize the robot. PID is a control loop feedback system related to minimizing error in real-time. This is done by continuously calculating the error value $e(\tau)$ which is the difference between the desired target and the measured value [41]. The PID equation can be seen below:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \quad (33)$$

There are three constants, proportional K_p , integral K_i , and derivative K_d . The proportional constant focuses on the current value of the error. The integral constant accounts for the remaining errors from before. Finally, the derivative constant focuses on estimating the future error based on the current rate of change [41]. These constants multiply their respective error in order to achieve a setpoint.

4.7 Cart-Pushing Studies

4.7.1 Stabilization Overview

Understanding the external forces caused by pushing a cart is an important challenge related to assisted walking via cart pushing. These external forces are applied directly to the robot's arms, so the robot must be able to compensate for these forces and mitigate the disturbance caused by the constant shift in its CoM [42]. A possible solution for this is by calculating the ZMP. If the ZMP shifts off the support polygon, it will cause the robot to fall. The support polygon is the horizontal region where the ZMP must lie over for the robot to be in static equilibrium. This idea is shown in Figure 4.4. Additionally, Figure 4.5 shows how this idea applies to a humanoid robot in motion. On the left-hand side, the ZMP is on the edge of the foot, causing it to start rotating. On the right-hand side, after the step has been taken, the ZMP is located over the support polygon in a balanced position.

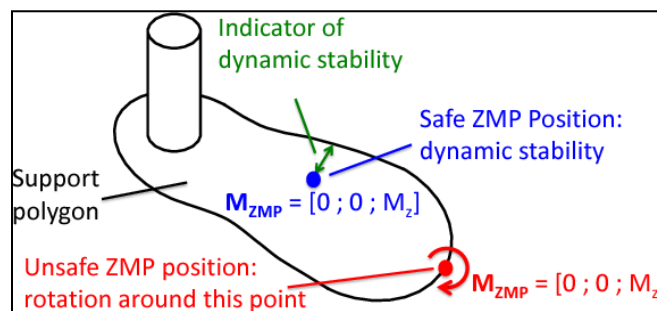


Figure 4.4: ZMP Position Example reproduced as is from [43]

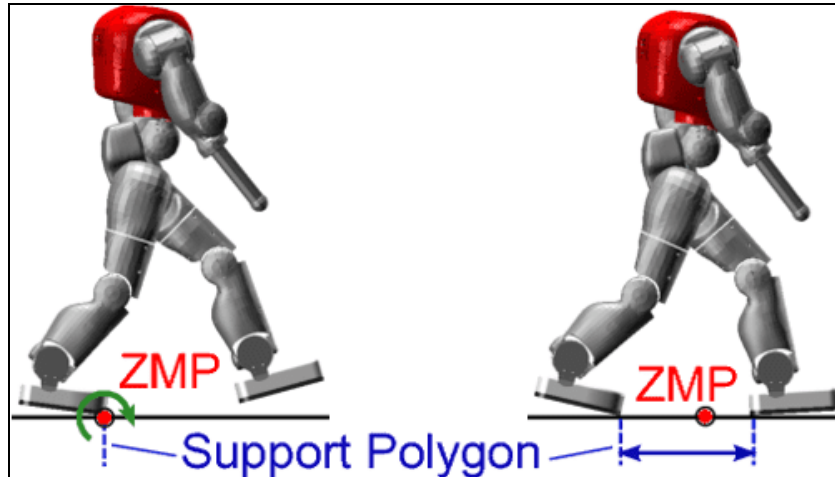


Figure 4.5: ZMP Example on a Humanoid Robot reproduced as is from [43]

4.7.2 Stabilization System and Calculations

Recent studies have explored this challenge with goals to improve a humanoid robot's manipulation abilities so it can maneuver utility carts in industrial scenarios and implement reliable arm compliance for stable bipedal walking. J.C. Vaz's "Material Handling by Humanoid Robot While Pushing Carts Using a Walking Pattern Based on Capture Point" study worked with the humanoid platform DRC-Hubo. DRC-Hubo has 32 DOF, weighs 80 kg, has a CoM height of 72.52 cm, a total height of 167 cm, an arm link length of 84.8 cm, and its wrists are equipped with force/torque sensors used to evaluate the force exerted by the cart it pushes (as shown in Figure 4.6). This study modified the classical Linear-Inverted Pendulum Model (LIPM) to account for external forces and combined arm compliance via a friction compensation method. This modification calculated and incorporated a capture point (CP); the CP is the point on the ground where the humanoid must step to come to a full rest. Simulations and real world experiments were performed to validate the dynamic model and the humanoid's overall mobility.



Figure 4.6: DRC-Hubo Humanoid Platform reproduced as is from [42]

To better understand LIPM with capture point dynamics for cart pushing, consider a unique point mass and a constant vertical CoM location where the acceleration of the CoM is given by:

$$x''_{CoM} = \frac{g}{z_c} (x_{CoM} - x_{ZMP}) + \frac{F_{cart}}{Mc} \quad (34)$$

The capture point (CP) is the point on the ground where the humanoid must step to come to a full rest:

$$CP_x = x_{CoM} + \frac{1}{\omega} x'_{CoM} \quad (35)$$

$$\text{where } \omega = \sqrt{\frac{g}{z_c}}$$

By taking the derivative of the second equation and combining it with the first:

$$CP'_x = \omega (x_{CoM} - x_{ZMP}) + \frac{F_{cart}}{\omega Mc} \quad (36)$$

Where g is gravitational acceleration, z_c is height of the CoM, x''_{CoM} is acceleration of the CoM, x_{ZMP} is the ZMP's location in the x direction, F_{cart} is force of the cart, M_c is the load mass on the cart.

Using the CP as state variable, the overall system dynamics are written as:

$$\begin{bmatrix} x'_{CoM} \\ CP'_x \end{bmatrix} = \begin{bmatrix} -\omega & \omega \\ 0 & \omega \end{bmatrix} \begin{bmatrix} x_{CoM} \\ CP_x \end{bmatrix} + \begin{bmatrix} 0 \\ -\omega \end{bmatrix} + \begin{bmatrix} 0 \\ \omega \end{bmatrix} \frac{F_{cart}}{Mc} \quad (37)$$

The target CP is given by:

$$CP_{x,k+1} = e^{\omega s_t} \left(CP_{x,k} + \frac{F_{cart}}{Mc} \right) + x_{ZMP,k} (1 - e^{\omega s_t}) \quad (38)$$

The Zero Moment Point is calculated as follows:

$$x_{ZMP,k} = \frac{CP_{x,k} - e^{\omega s_t} \left(CP_{x,k} + \frac{F_{cart}}{Mc} \right)}{1 - e^{\omega s_t}} \quad (39)$$

Where $CP_{x,k}$ is the ongoing CP, $x_{ZMP,k}$ is the current ZMP in k , and s_t is the step time.

The force of the cart is obtained from force/torque sensor data, and the model is validated in MATLAB simulations. Figure 4.7 shows the forces generated by the cart and the mass.

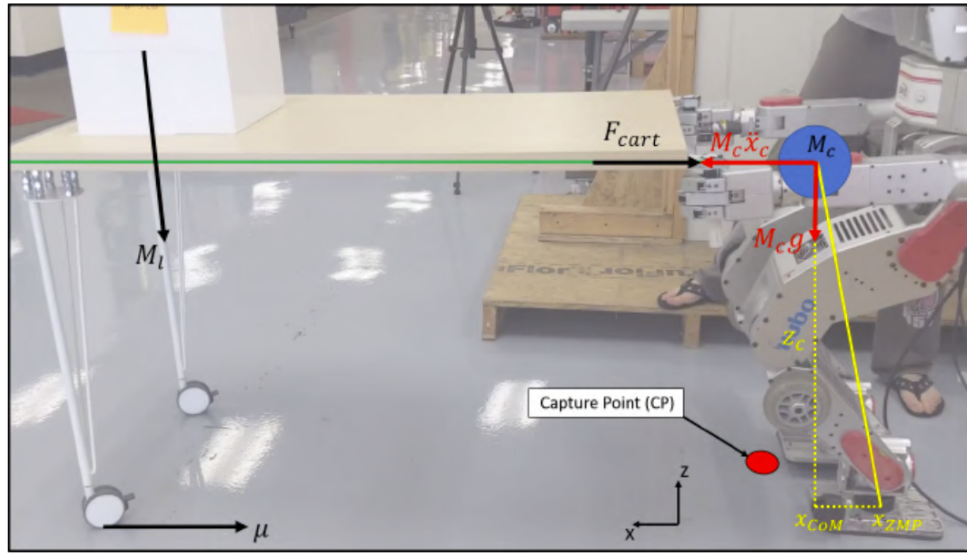


Figure 4.7: LIPM with Capture Point Dynamics for Cart Pushing reproduced as is from [42]

For the arm compliance control, a friction compensation method was used. A block diagram of this system is shown in Figure 4.8, where x_{wrc} is the current position of the wrist and x_{wrd} is its desired position.

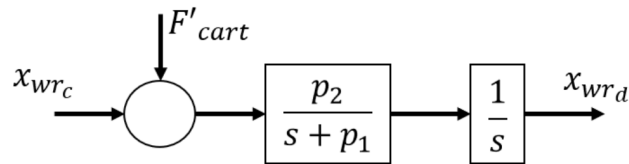


Figure 4.8: Reduced Block Diagram of the Friction Compensation System

For simplicity, F_{cart} , p_1 and p_2 are given as:

$$F'_{cart} = \frac{F_{cart}}{\tau_c K_a} (x) \quad p_1 = \frac{\mu + \tau_c f_c}{J} (x) \quad p_2 = \frac{\tau_c K_a}{J} (x) \quad (40)$$

Where τ_c is the torque constant, K_a is amplitude gain, μ is coefficient of friction, f_c is the arm moment of inertia, and J is the wrist moment of inertia. Figure 4.9 shows the overall system integration.

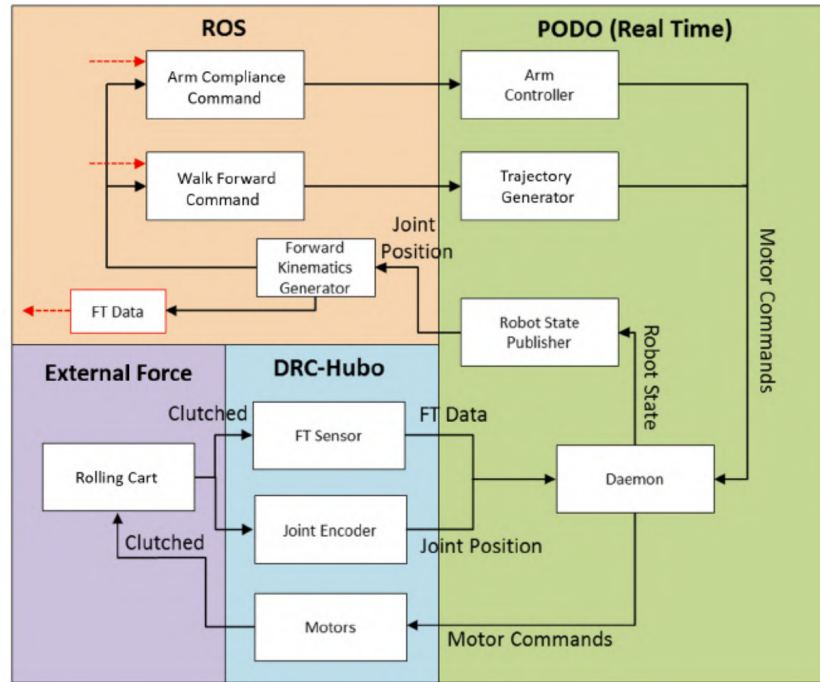


Figure 4.9: System Integration reproduced as is from [42]

Once the rolling cart is securely coupled, DRC-Hubo can read force/torque data through corresponding sensors, and the joint encoder will monitor the actuator's real positions. PODO (software that DRC-Hubo uses to communicate with the programs, sensors, simulator, and user) is the robot's communication software and is responsible for sending motor commands to the controllers. A robot state publisher is responsible for the exact joint position during operation. The system integration, in Figure 4.7.6, is represented by four main sectors; PODO, Humanoid, External Force and ROS (Robot Operating System) [42].

During experimentation, the robot is initially set in a “walk ready” position to check all FT sensor statuses before initiating any movement. Then, the pushing experiment is initiated which has three layers of operation. First, Hubo's main computer runs the modified pushing algorithm to generate the appropriate walking trajectory. Second, the main computer constantly monitors the FT sensor data from both wrist and feet. Third, the operator must initiate the experiment through the Graphical User Interface (GUI) after the insertion of certain parameters (step length, number of steps, and weight of load). Finally, motion can commence.

4.7.3 Cart-Pushing Results

The results of this study showed that the rolling cart has an impact on the walking gait due to the downward force on the wrists. Additionally, the ZMP error, the root mean square error (RMSE) between the reference and actual X-ZMP, was measured for each case. A loadless rolling cart yielded the least error followed by a 25.22% error increase while carrying 11 kg. In contrast, for the utility cart the error increases only 10.69% between a loadless and 11 kg load due to the geometric contact favorability of a utility cart (four contact points [42]). Table 4.3 shows the ZMP error or RMSE for each case. Both the rolling cart and utility cart can allow stable walking, even when pushing a heavy load. More than 30 trials were performed to validate the approach, and all but two trials yielded satisfactory results. These results are shown in Figure 4.10. The proposed method can be adapted to different carts, and the pushing motion can be performed during stable walking motion.

Table 4.3: RMSE of Experimental X-ZMP vs. Calculated X-ZMP reproduced as is [42]

Cart Type	Load [kg]	RMSE
Rolling	0	0.0252
Rolling	11.7	0.0337
Utility	0	0.0284
Utility	11.7	0.0318

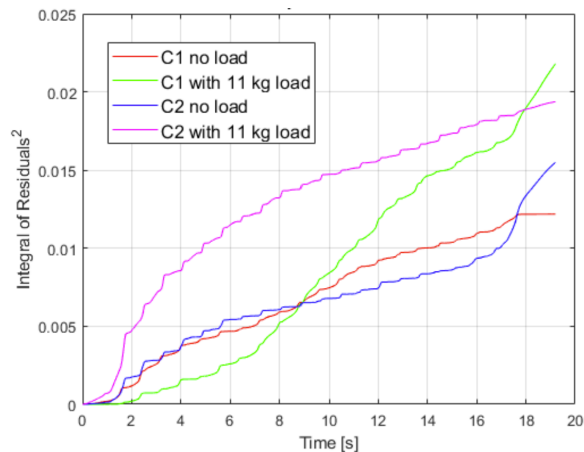


Figure 4.10: Residual Squared Sum Over Time for Rolling Cart “C1”, and Utility Cart “C2” with and Without Load reproduced as is from [42]

The key takeaways from this study for exploring assisted walking via cart pushing are the three layers of operation (running algorithm to generate walking trajectory, constantly monitoring sensor data from wrist and feet, and initiating experiment through GUI after inserting parameters), placing FT sensors in the wrists and feet, and possibly using a LIPM with capture point for cart pushing and walking calculations. This study also emphasized the importance of considering the feedback provided from the cart while pushing it and how to adjust for it; this study used an arm compliance control with friction compensation method to this end.

4.8 UI Prototyping

In order to create a well-designed UI, there are many factors that need to be considered. Most importantly, the UI needs to allow communication between the machine and the user. It needs to allow the user to perform an action, such as pressing a button, and see an immediate response from the machine, such as a robot moving.

In order to judge a UI’s quality, it must be evaluated. One way to evaluate a IU is heuristic evaluation. Jakob Nielsen’s ten heuristics are the most widely used when considering an application[43]. These ten heuristics are used during prototyping to ensure useability and remove problems[44]. The benefits of this technique are its ease of identifying issues, ease of implementing. The ten heuristics are as follows:

1. Visibility of System Status
2. System Matches Real World
3. User Control and Freedom
4. Consistency and Standards
5. Error Prevention
6. Recognition Rather than Recall
7. Flexibility and Efficient of use
8. Aesthetic and Minimalist Design
9. Recognize, Diagnose, and Recover
10. Help and Documentation

The first is known as visibility of system status. This implies that the system tells the user what is happening and displays results. For example, a loading screen tells the user that the page is loading as well as when loading is complete. The second is that the system matches the real world. This means that if a user is trying to delete something, the language of “delete” is used consistently instead of “eject” or something different. Third is user control and freedom. User control and freedom allows the user to navigate through the application in any way they please. In this step, evaluators look especially for back and undo buttons, which are essential to prevent users from performing unwanted actions. Fourth is consistency and standards. This means using commonly understood words such as “save”, “edit”, etc, and features like having the same font throughout the interface. Fifth is error prevention. This means that steps are taken to prevent errors before they are allowed to occur. For example, if a user was expected to enter a date, instead of having an open input box, the application would have designated spots for the day, month, and year. Sixth is recognition rather than recall. This simply means reminding users of what their options are rather than expecting memorization. Seventh is flexibility and efficiency of use. This consists of allowing users to use an application with greater speed and effectiveness as they learn it. Eighth is aesthetic and minimalist design. This implies that visuals are not cluttered and only important information is shown. Ninth is helping users recognize, diagnose and recover from errors. This means that the application helps users diagnose errors and provides a data-driven solution. Lastly, help and documentation is essential. Allowing the user to click on a help button or having a search bar is important to creating a well-designed UI [45].

This evaluation is simple to perform. Members of the development team go through each section of the prototyped application. In each portion, the ten heuristics are considered, and comments are written about each heuristic. These comments can be any observations such as, saying that a section has aesthetic and minimalist design and does not need to be edited in that way. The comments can also provide constructive criticism, like saying that there is no way to recover from errors in a section. Then, the results are interpreted and addressed by the team.

There are some limitations to heuristic evaluation. One of the most prominent problems is that it identifies problems, but does not suggest ways to fix them [46]. While it is helpful to identify problems, developers still have to come up with their own solutions. Another issue is that developers performing these evaluations are limited to the mindset that they are in. Additionally, heuristic evaluations are not as defined and controlled as an evaluation like a user test. With a user test, there is a clear setup, procedure, and evaluation technique used. With this heuristic evaluation, there is no way to validate the procedure. However, when doing preliminary tests, a heuristic test can provide designers with ways to improve in the prototyping stage before bringing in outside users [47].

4.9 Simulation

Simulation is an easy way to test robot functionality more easily and safely than physical tests. In the field of robotics, the main use of a simulation is to test a robot's movement in a given environment [48]. These simulations can generally be modified along a spectrum of accuracy, from complete abstraction with only basic physics, to as close to reality as possible [48]. Virtual simulations can be used to test the safety of robotic surgery tools or underwater vehicles [49, 50]. As such, when employed correctly, they can play a key role in validating development decisions and early identification of problems. Simulation testing can concern one or both of the two broad areas of concern on a robot: its physical interaction with its environment, and the efficacy of its control code. Advanced simulation software such as V-REP and Gazebo allow for the integration of 3-D models of physical features with an API allowing for the implementation of original control code, rather than choosing one or the other [51]. In contrast, Blender is designed for much more general simulation of physics and movement, rather than robots in specific [51]. As a result, it lacks many of these specialized features that allow for convenient testing pipelines in industry-specific software. This allows for comprehensive and simultaneous testing of both

physical and software modifications made to the robot, greatly increasing the efficiency of testing and validation.

5. Methodology

This chapter discusses the process of first understanding the previous work of this continued project, choosing new motors and sensors, and developing a simulation. Then, the process towards developing standing and assisted walking in simulation and then the real world is outlined. Lastly, the process for developing a new user-friendly interface is discussed.

5.1 A Term

5.1.1 Understanding Koalby's Initial Status

Last year's team successfully created the humanoid robot, Koalby. At the outset of our team's work, we aimed to fully understand how exactly Koalby functioned, what needed improvement, and what could be added. The electrical and electronic components were examined for damage and weaknesses. These tests were conducted continuously throughout the development of the robot, especially when it was not functioning as expected. Any sources of possible or real problems for robot operation were recorded and compiled into a troubleshooting document for future reference. These results can be found in Chapter 6. This allowed for greater efficiency in fixing repeated problems, as well as identifying which areas to prioritize for improvement, as parts that broke most frequently could be considered to be the most important to prevent breaking in the future. Overall, this method was used to ensure that Koalby was consistently functional so that thorough tests could be conducted without results being confounded by extraneous factors.

Koalby's control software was also examined for areas of improvement. In particular, the team noted pieces of code that were difficult to understand or did not function consistently. By reducing the time spent in the lab repairing broken pieces, more time could be allotted to conducting and iterating tests, which increased their effectiveness and efficiency. In addition, we identified portions of code that were left incomplete by last year's team, and developed a plan for either completing their development or discarding unnecessary portions. Targeting unclear or incomplete functions also served to ensure that any continuation of this project would have an

easier time onboarding to testing and development. These software improvements are discussed in detail in Chapter 12.

5.1.2 Code Refactoring

In order for the code to incorporate every aspect we wanted to implement, it needed to be refactored. While refactoring, ideas such as processing time and overall functionality were considered. We also assessed consistency, readability, and useability. This refactoring started in A term, and was carried out the whole year as new functionalities were added.

5.2 B Term

5.2.1 Forward and Inverse Kinematics

With the goal of upgrading Koalby's movement to be more fluid and accessible to users as well as to move the arm to objects to pick them up once the different sensors are implemented, its arm and leg were analyzed as kinematic chains in order to calculate their forward and inverse kinematics. The Denavit-Hartenberg Method was used for forward kinematics calculations for both limbs and DH tables were made for each of them. The inverse kinematics of the leg were calculated using geometric analysis of an open-loop model of the chain, but the inverse kinematics of the arm were found using the Product of Exponentials method due to the added complexity of Koalby's shoulder joint, as it had three motors in a condensed area actuating different movements. These calculations were then implemented into Koalby's software and testing was done using dummy values in the code to validate the calculations. This process is more thoroughly explained in Chapter 7.

5.2.2 Simulation Software

In order to conduct thorough tests of the robot, the team needed to create a simulated version of it. This allowed for more convenient and safe testing of Koalby's functions, allowing for tests to be run at home or when the robot is otherwise unavailable and eliminating the risk of damaging the robot if a test goes wrong. In order to ensure that the simulation suits the needs of the group, a set of criteria was created for selecting a software in which to model the robot. These criteria and their results are discussed in Chapter 10. These criteria were cross-platform

compatibility to allow the entire team to work together easily, compatibility with the existing robot control code, and the ability to simulate the sensors which we planned to include on the physical robot. Various simulation software, including Blender, Gazebo, and CoppeliaSim were assessed both by reading documentation and by running test simulations, and ranked by how well they fit the necessary criteria, again discussed in detail in Chapter 10. The software determined to be the best fit would then be used throughout the rest of development for testing and verification of new features.

5.2.3 Building Koalby in Simulation

In order to simulate Koalby, he needed to be built in simulation. A URDF file of Koalby's 3D model was initially used to generate an accurate model of Koalby in simulation. However, due to numerous errors in transferring that file into our simulation software, we ended up using CoppeliaSim's built-in ability to draw and add shapes and connect them by placing joints between two shapes. Using this functionality, we created Koalby out of many rectangular prisms connected by joints following the robot's actual dimensions as accurately as we could for use in simulation testing. This is elaborated on in Chapter 10.1.

5.2.4 Motors

New motors were considered to standardize everything to one brand. Standardizing to one brand would improve the coding time by only requiring one motor library and eliminating the need for additional hardware. When comparing different motors, the functionality differences (i.e. communication speed and resolution) between Dynamixel and Herkulex motors were considered.

5.2.5 Sensors

In order to stabilize the humanoid, we needed to implement sensors into the robot for feedback control. We examined multiple sensors mainly through literary research and then through testing. This research and experimentation can be reviewed in Chapters 4.4 and 9.3. When considering the different sensors, two main aspects were examined: aiding in stabilized standing and sensors that would provide more autonomy in the environment.

5.2.6 Code Architecture Evaluation

In order to incorporate the user interface into the existing application, the project architecture needed to be evaluated. We needed a way to connect to the user interface wirelessly while still being able to control the robot and connect to the Arduino.

5.3 C Term

5.3.1 Trajectory Planning

For enabling Koalby to walk, we first had to choose a suitable trajectory generation method to control the leg's position and velocity over time. This led to the team analyzing the benefits and drawbacks of linear, cubic, and quintic polynomials through research in order to determine which would best suit our needs. Once this was decided, the calculations necessary for trajectory generation were implemented in Koalby's software. Testing was then done in simulation with this trajectory planning functionality by feeding a CSV file different joint positions to determine the optimal step for Koalby while pushing the cart. This is explored further in Chapter 8: Trajectory Planning.

5.3.2 Determine the Position and Motor Angles

A new initialization position was considered to find where Koalby stood most stable. Having this position at initialization would allow for consistent standing results and easier testing of the robot standing stable. Each individual motor position needed to be considered in order to make the whole robot stable. By initializing the robot in various test positions and recording how long it was able to stand stably in simulation and in reality, we determined the optimal initialization position.

5.3.3 Gather IMU Readings and Stabilized Controls

The Arduino IDE was utilized to conduct a series of tests on the IMU, with a focus on gathering readings at varying speeds and motions. By examining various metrics, such as noise and bias levels, we were able to determine the most suitable raw data for use in the implementation code. These tests facilitated a better understanding of the IMU and enabled the

development of a code framework that could be readily integrated into the Python code. Once the most stable motor positions were established, the IMU was placed near the center of mass and the pitch and roll values were collected. These values were then created as the target motor positions and a PI controller was implemented in order to keep the robot stabilized in simulation. The Proportional and Integral values were tested until the most stable constants were found.

5.3.4 Choose a Filter

Due to the noise of an IMU a filter needed to be applied to the IMU readings in order to stabilize the robot. Two filters that were compared and considered were the complementary filter and the Kalman filter. The main criteria used to determine which filter to use came down to computation time and accuracy. The comparison between these two filters can be reviewed in Chapter 4.5.

5.3.5 UI Prototyping

Different UI designs needed to be considered before making the user interface. We wanted full robot control from the UI, so we needed a way to incorporate everything while keeping the design as simple as possible. We also wanted it to be consistent throughout, so having a pre-designed plan allowed for this and also made it easier to implement in the code. When this was complete, a heuristic evaluation was performed and informal feedback was gathered to test its functionality and useability.

5.4 D Term

5.4.1 Testing Stability

Tests were conducted to determine what was considered a stable stance for Koalby. We first tested in simulation, then on the real robot. The idea behind this was that in the simulation, there are fewer outside factors, such as loose screws or wires, that needed to be considered. Additionally, the consequences of failure were a lot less in the simulation because Koalby could topple over and fail without risk of parts or wires breaking. From there, using control code tested in the simulation, we could move to the real world and make fewer tweaks.

The first set of tests, conducted in simulation, included testing if Koalby could stand without assistance and if Koalby could stand with assistance only from a held cart. After these tests were conducted, a new set of tests were done to determine if Koalby could perform basic actions such as waving and handshaking while stable. Lastly, once these were all conducted in the simulation, they were tested on the real robot.

5.4.2 UI Testing

After various tests were performed in the simulation and the real world from the text based code output, these same functionalities were tested from the UI.

6. Evaluating Koalby

6.1 Testing Electronics

When the team first received the robot, it had sustained significant damage from sitting in storage. Several of the 3D-printed pieces were chipped or completely broken, and many wires were frayed or snapped. This meant that a significant portion of the beginning stages of the project was dedicated to restoring Koalby to its original working order. The most significant problems and solutions are explored below, while a more in-depth listing of every issue encountered by the team can be found in Appendix C.

It became quickly apparent that the most vulnerable components were pieces that connected the different limbs. In particular, the areas connecting the legs to the torso faced large amounts of stress and broke frequently. The pelvis, shown in Figure 6.1, had to be replaced on three separate occasions during the first four months of working on Koalby.

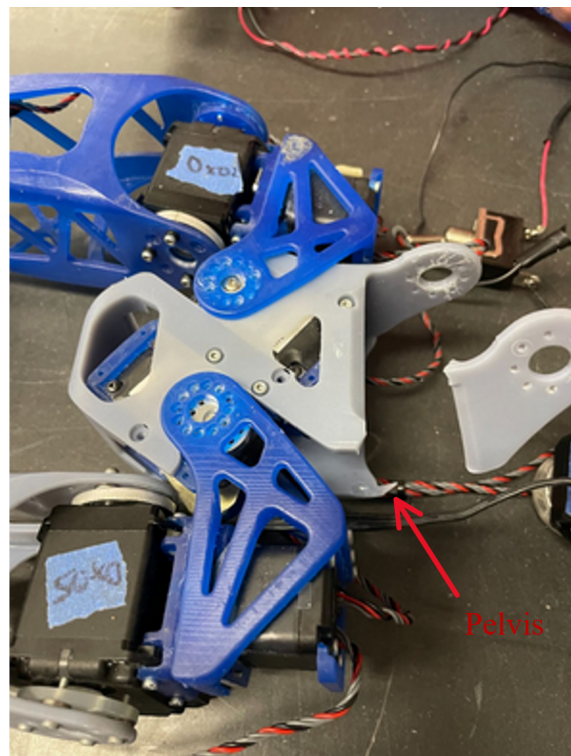


Figure 6.1: Koalby's Broken Pelvis

While the redesign of these pieces was being performed by the design team, we worked to mitigate these issues as best we could. This included solutions such as changing the initialization of the robot's motors to move at a half of their original speed and using a frame to hold Koalby for testing. Slower movements reduced the chance of the robot damaging itself or pulling wires loose, while the frame reduced the initial stress placed on the motors due to the removal of the robot's body weight from consideration. We also worked to improve Koalby's wiring, which also regularly experienced breakage. Additionally, new and higher quality parts, such as switches and connectors were purchased. The primary improvement was replacing solid core wiring with threaded wires of the same gauge. These threaded wires were more flexible and broke less frequently than the solid cores that they replaced.

The two most common sources of problems on the electronics side were the Arduino board and the primary power switch. The Arduino largely made use of jumper cables, and as a result its connections were prone to being disconnected by the robot's movements. The primary power switch was suspended off of the main body of the robot. On rare occasions, these small issues of electrical disconnection escalated into a more dangerous shorting of the power supply, causing smoking and permanent wire damage. Figure 6.2 shows an example of smoke rising up from underneath the robot due to a shorted connection.

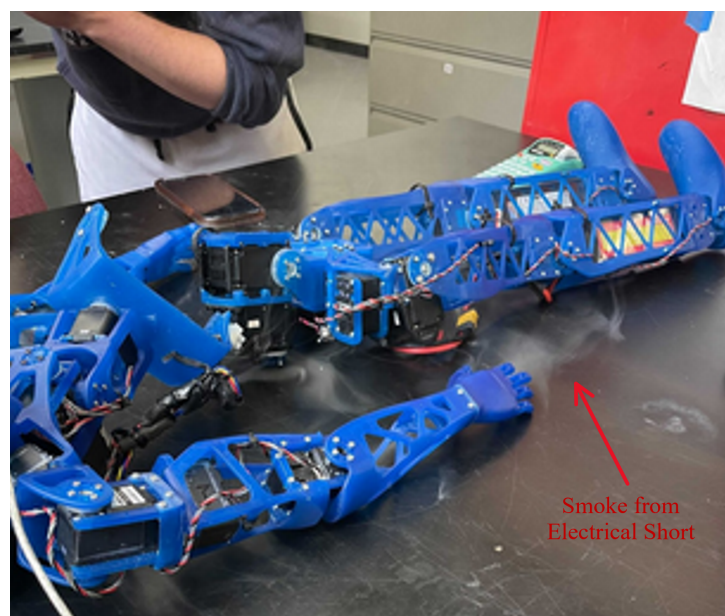


Figure 6.2: Electrical Short

As a result, the team engaged in frequent re-wiring of various components of the system, as well as replacing and re-soldering the main power switch. It was also found that reducing the range of movement of the wires and other electrical components subsequently reduced the opportunity for these components to become damaged by jostling or tension. As such, they were secured in place as much as possible, using electrical tape to group wires together into bundles and zip-ties to connect components to stable portions of the body such as the back of the torso. Figure 6.3 illustrates a collection of wires with all of these protections applied.

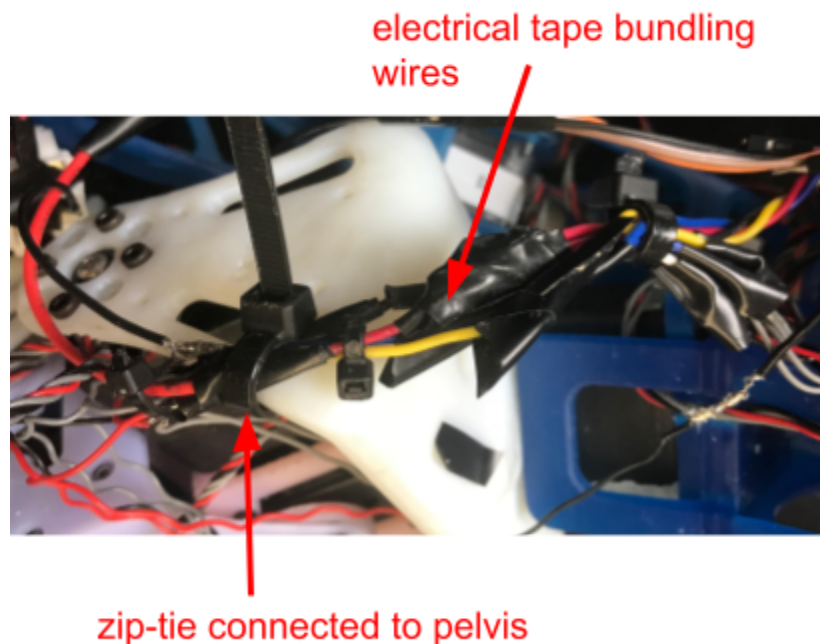


Figure 6.3: Restricted Wires

The Arduino Mega control board was bolted to the side of the head and the various sensors were given custom mounts to be held in place. The Arduino's mount can be seen in Figure 6.4 below.

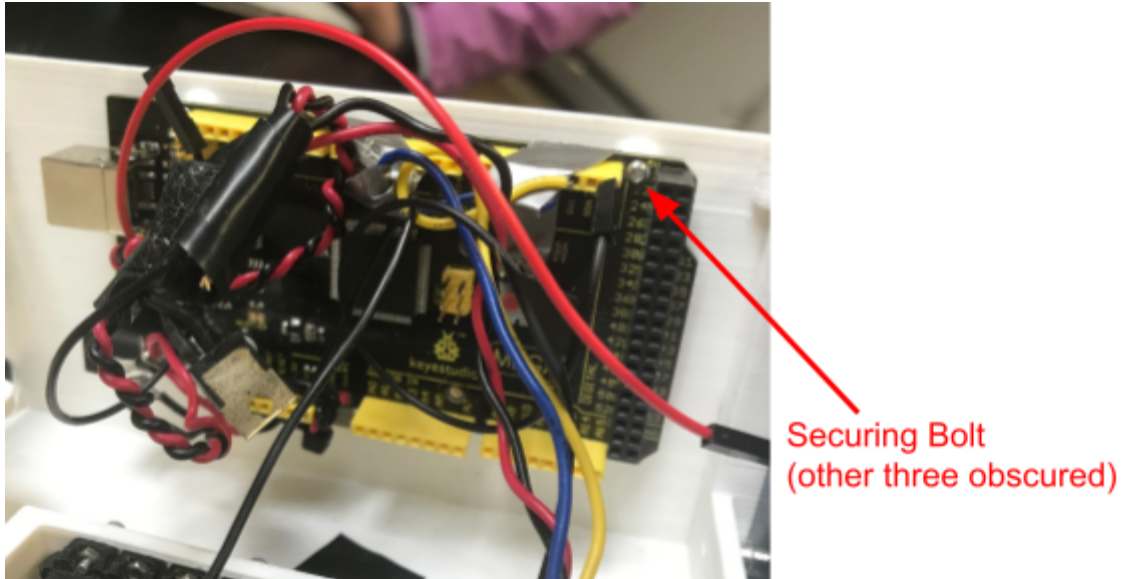


Figure 6.4: Bolt-Mounted Arduino

The final electronic improvement we made was to implement a system for tracking the voltages of all three batteries that powered the robot. We found that the stored potential of these batteries had a significant impact on both the speed and load-bearing capacity of Koalby's motors, making it worthwhile to keep an accurate record of exactly when each battery needed to be charged.

6.2 Testing Software

The software to control the robot developed by last year's team consisted of two portions: Arduino code to directly change the positions of the motors, and Python code to make decisions about what movements to make, when to make them, and pass on instructions to the Arduino. Koalby's code structure is explored in more depth in Chapter 2.5. As with the hardware, the team documented the issues encountered when trying to run this software, along with their solutions. Often, these problems were determined to have causes on the hardware side, such as the aforementioned disconnections of wires from the Arduino. The team also worked to resolve inconsistencies in the coding and documentation of the software. Many functions worked but it was not always clear what they accomplished, so we tested and documented these functions as thoroughly as possible. This consisted of running functions one at a time and recording the results. When the results were not immediately clear, the debugging feature available in the

Pycharm IDE was used to go through the code's actions step-by-step for an in-depth understanding of what it aimed to accomplish. The largest hurdle in interpreting the software was the organization of the robot's motors. Both the Python and Arduino code kept track of the motors by numbering them, but the two codebases used completely separate systems for numbering motors, so motor IDs changed depending on the level of abstraction of the code. The team created and updated as necessary a system for identifying motors and their corresponding IDs from one set of code to another. The diagram seen in Figure 6.3 was created, which displayed each motor's physical position on the robot as well as how it is identified in both Arduino and Python code. This diagram was updated as new motors were introduced into the system, and is seen in its most recent form below. This allowed for more effective assessment and modification of individual motors when such actions became necessary.

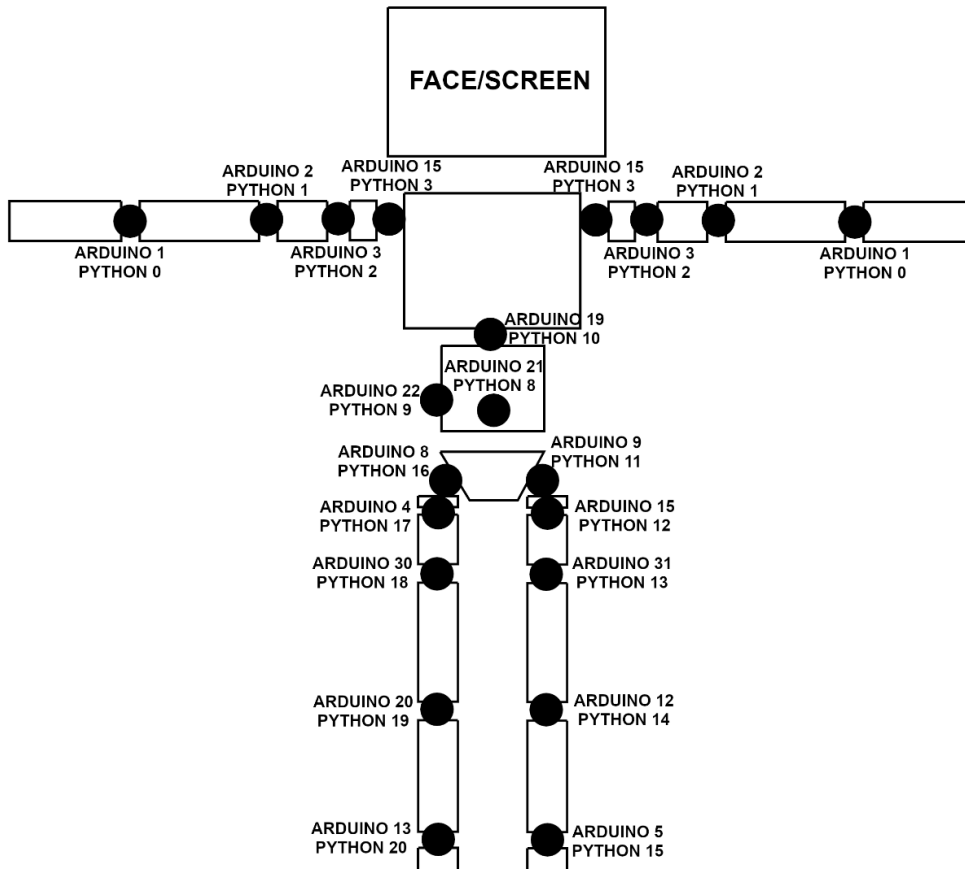


Figure 6.3: Koalby's Motor IDs

As indicated in the methodology chapter, both the physical and software test results were thoroughly documented throughout the term, the complete results of which can be found in Appendix C. The most common problems arose from motors either reaching their torque limits and giving out or not receiving proper serial communication. Over-torqued motors could be prevented by removing physical obstacles in their paths, the aforementioned support frame, and recharging the batteries, as reduced available power weakened the motors. The serial communication issue most often was the result of the wires originating in the Arduino popping out of their ports. These would simply have to be plugged in, and were eventually secured in place with tape to reduce the frequency of these disconnections. This continual documentation allowed for streamlined problem-solving for repeated problems as well as identifying these problems and prioritizing them for more permanent solutions. In addition, this document could serve as a useful starting point for future project teams to prevent repeated fixing across iterations.

7. Kinematics

Forward and inverse kinematics were calculated for Koalby's leg and arm through use of the Denavit-Hartenberg and Product of Exponentials methods. These calculations were implemented into Koalby's control software.

The home configuration of the leg was assumed to be at the position where every link points downward, indicating that Koalby is standing straight up and mimicking the initialization position we made him start at where every joint is at its zero position. The open-polygon model of this configuration is shown in Figure 7.1. The DH table for this configuration is shown in Table 7.1.

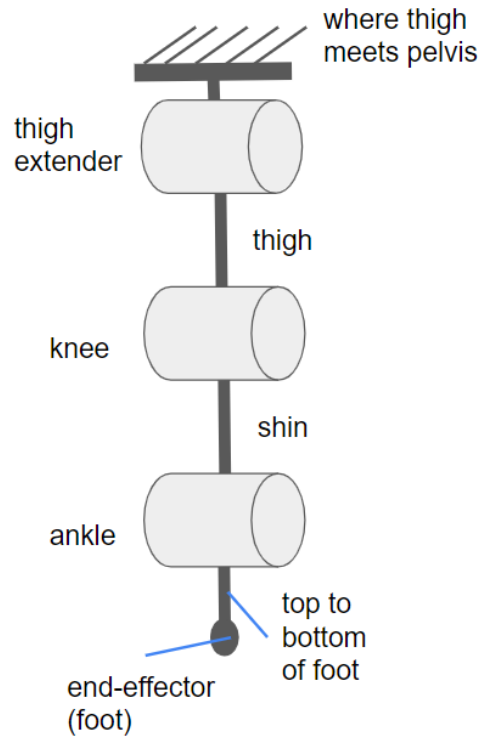


Figure 7.1: The Open-Polygon Model of Koalby's Leg

Table 7.1: The DH Table for the Open-Polygon Configuration of Koalby's Leg Shown in Figure

7.1

Link	θ (degrees)	d (mm)	a (mm)	α (degrees)
1	θ_1^*	0	187.325	0
2	θ_2^*	0	212.725	0
3	θ_3^*	0	34.925	0

The core equations used for the inverse kinematics calculations were found using a geometric analysis of the leg kinematic chain modeled as shown in Figure 7.2. This was done primarily through the use of trigonometry and drawing triangles composed of parts of the chain to determine equations (50), (51), and (52), representing the angular position value of each joint in any leg configuration. The equations for the calculation of the three different joint values can

be seen in the same image, where ‘x,’ ‘y,’ and ‘z’ are the Cartesian coordinates of the end effector in the task space.

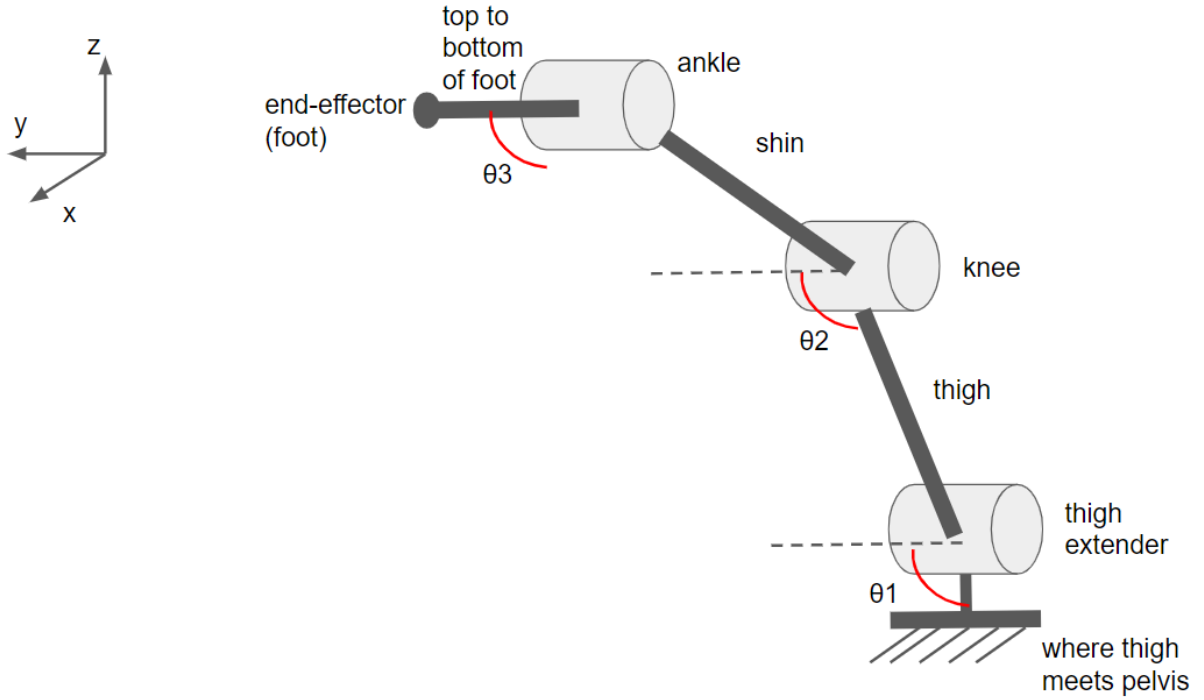


Figure 7.2: Open-Polygon Model of Koalby's Leg

$$r = \sqrt{x^2 + y^2} \quad (41)$$

$$s = z - l_1 \quad (42)$$

$$d = x/r \quad (43)$$

$$e = (l_2^2 + r^2 + s^2 - l_3^2)/(2 * l_2 * \sqrt{r^2 + s^2}) \quad (44)$$

$$f = \sqrt{1 - e^2} \quad (45)$$

$$g = (l_3^2 + l_2^2 - (r^2 + s^2))/(2 * l_2 * l_3) \quad (46)$$

$$h = \sqrt{1 - g^2} \quad (47)$$

$$\alpha = \text{atan2}(s, r) \quad (48)$$

$$\beta = \text{atan2}(e, f) \quad (49)$$

$$\theta_1 = \text{atan2}(\sqrt{1 - d^2}, d) \quad (50)$$

$$\theta_2 = -(\alpha - \beta) \quad (51)$$

$$\theta_3 = - (\text{atan2}(h, g) - \pi/2) \quad (52)$$

Following the calculations for the forward and inverse kinematics of the leg, the DH parameters and equations that were calculated were hardcoded into Koalby's software. This was done to expedite and automate any further kinematics calculations and, at least for forward kinematics, to read the robot's current joint angle values and calculate where the position of the foot is at those positions. Unlike forward kinematics, the position of the end-effector in Cartesian space needed to be provided by the user, but the joint positions that correspond to the given position were easily calculable through the code. In the software, the forward kinematics functionality reads the different angle values of the joints and creates a 4x4 transformation matrix representing the transformation from the base of the thigh to the bottom of the foot, with the Cartesian position of the bottom of the foot found in this matrix. The inverse kinematics functionality takes a desired Cartesian position inputted by a user and produces the joint values that correspond to this position using the equations derived in equations (41) through (43). A flowchart of the kinematics functionality in software for the leg can be viewed in Figures 7.3 and 7.4, and all of it was put in place to allow a user to move the foot to a specific Cartesian position or to specific joint positions by just telling the code the desired position and letting the various functions do the calculations for them.

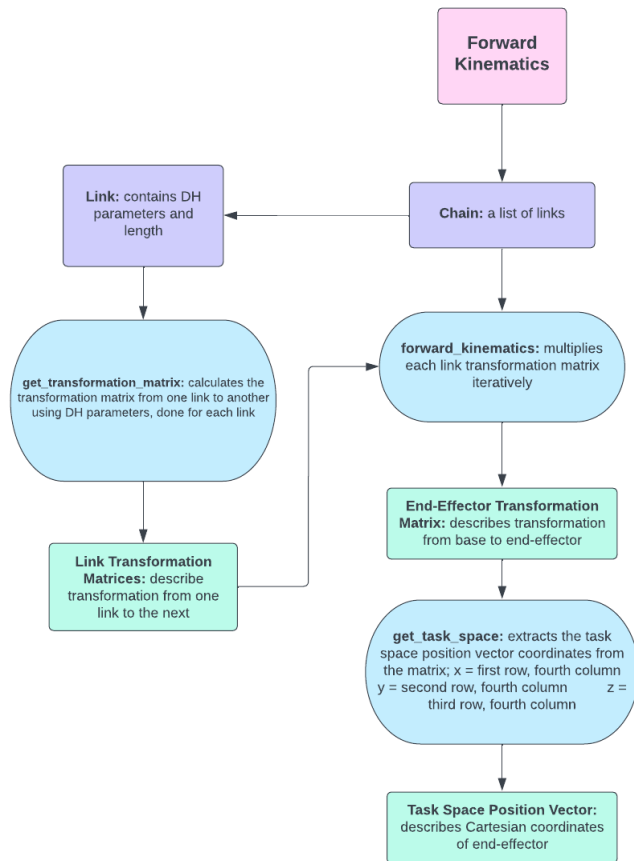


Figure 7.3: Flowchart of Forward Kinematics Code Functionality in Koalby's Software

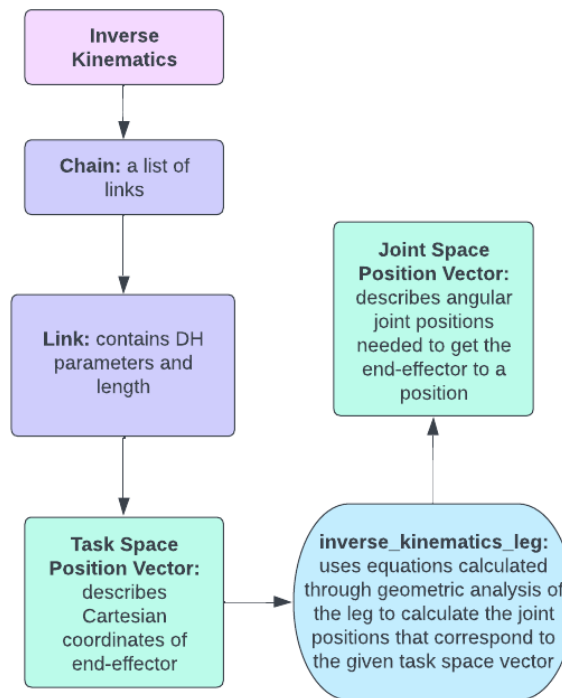


Figure 7.4: Flowchart of Inverse Kinematics Code Functionality in Koalby's Software

The forward kinematics of the arm were implemented in a similar way to the leg as the DH method was used for the calculations. The home configuration of the arm was modeled with the arm straight out at each motor's zero position, and this configuration is shown in Figures 7.5 and 7.6. The DH table created through analyzing this configuration is shown in Table 7.2. Each joint was established as the length between the end of the previous motor to the beginning of the following one instead of the printed parts.

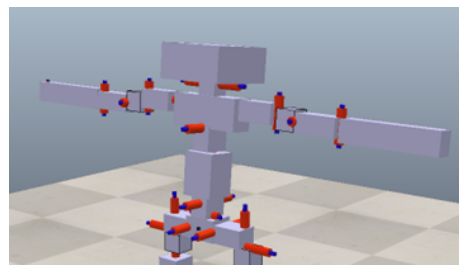


Figure 7.5: Koalby's Arm Straight in Zero Position in Simulation

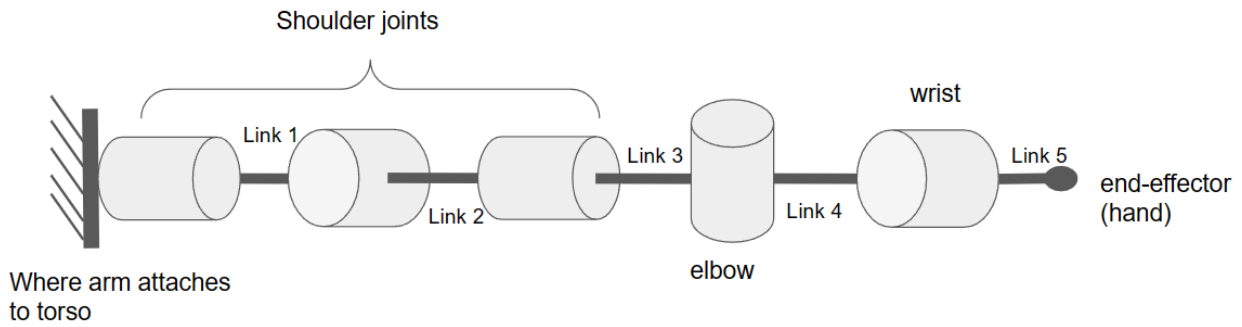


Figure 7.6: The open-polygon model of Koalby's arm

Table 7.2. The DH Table for the Open-Polygon Configuration of Koalby's Arm Shown in Figure 7.6.

Link	θ (degrees)	d (mm)	a (mm)	α (degrees)
1	θ_1^*	50.4	0	-90
2	θ_2^*	0	0	+90
3	$\theta_3^* + 90$	158.81	0	+90
4	$\theta_4^* + 90$	0	92.03	+90
5	θ_5^*	0	86.76	0

The inverse kinematics of the arm were found using the product of exponentials method and MATLAB code which takes in the homogeneous transformation matrix M and the Screw matrix and the starting joint position of $q[0]$, which is a 1×5 matrix, and the target end position matrix, which is a 4×4 matrix, to get joint angle positions. This performs forward and inverse kinematics based on the DH table found above and jacobian as a way to control the velocity of the end effector.

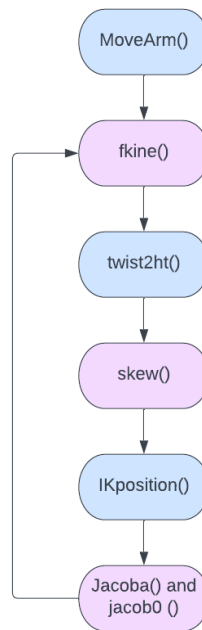


Figure 7.7: Flowchart of Arm MATLAB Code

Due to time constraints the code was not tested in simulation or on the real robot. However the validity of the calculations were tested in MATLAB. The inverse kinematics code was validated by running the forward kinematics on the final position result of the inverse kinematics function.

8. Trajectory Planning

The first step of the trajectory planning process was for us to choose which trajectory polynomial we should use for Koalby's walk cycle, either linear, cubic, or quintic. A linear polynomial was quickly rejected as it only allows for control over position constraints, which could lead to unpredictable leg velocities that could be damaging to both the robot itself and other people. This then left cubic and quintic trajectories to decide between. As mentioned in Chapter 4.2, cubic trajectories allow for control of position and velocity constraints, while quintic trajectories allow for additional control of acceleration constraints with slightly added complexity. Our team figured that the most stable and consistent step would be one where the leg moves at a constant velocity with no acceleration, so we chose to use a cubic trajectory polynomial as there was no need for the acceleration control that quintic polynomials offered. However, we decided to make it easy to implement quintic trajectory planning into the software if future teams desired to by generalizing most of the functions to work regardless of trajectory planning method chosen.

Upon choosing a cubic polynomial as our trajectory calculation method, we implemented the trajectory planning into Koalby's software. A flowchart of how the trajectory planning functions is shown in Figure 8.1.

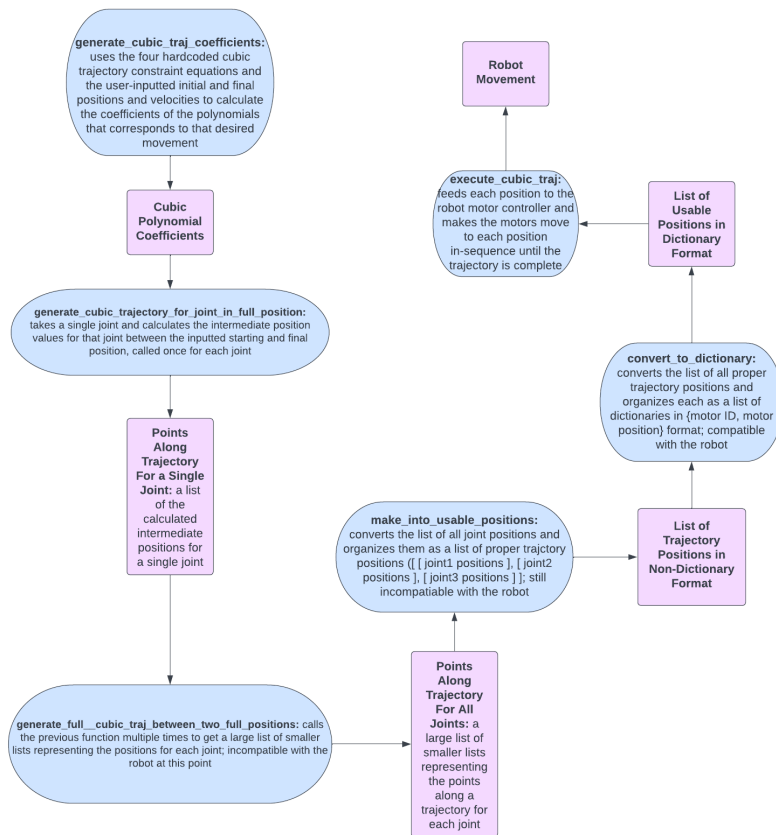


Figure 8.1: Flowchart of Trajectory Planning Code Functionality in Koalby's Software

In essence, the software is fed the joint angle positions of each of the leg motors, as well as desired input and output times and velocities specified by the user and calculates a series of intermediate points between the robot's current position and that final desired position. Specifically for walking, the software would read different angle positions representing an initial and final leg position from a CSV file, convert them to a dictionary in $\{motor\ id: motor\ position\}$ format, and do numerous calculations to interpolate between these two positions. The starting and final positions along with all of the generated points in-between would be sent back to the robot in the same dictionary format and the robot would then move through these points. An example of the interpolated points calculated is shown in Figure 8.2.

```
{18: 116, 19: 14, 20: -12}  
{18: 116.0, 19: 13.55775, 20: -11.99275}  
{18: 116.0, 19: 12.292, 20: -11.972000000000001}  
{18: 116.0, 19: 10.294249999999998, 20: -11.93925}  
{18: 116.0, 19: 7.655999999999999, 20: -11.896}  
{18: 116.0, 19: 4.46875, 20: -11.84375}  
{18: 116.0, 19: 0.8240000000000007, 20: -11.784}  
{18: 116.0, 19: -3.186749999999998, 20: -11.71825}  
{18: 116.0, 19: -7.471999999999996, 20: -11.648}
```

Figure 8.2: Example of Software-Generated Points Between a Beginning and Start Trajectory in `{motor id: motor position}` Format to Be Fed Back to the Robot for Movement

The amount of intermediate positions generated is controlled by a timing parameter in the software that determines the delay between point calculations along the desired trajectory. For example, if the timing parameter was set to 1 second, a new intermediate point would be generated by incrementing the time step by 1 until reaching the final inputted desired position in the trajectory. Despite being built specifically for getting Koalby to perform a walking motion, this trajectory code can also be used with other limbs, like the arms, to facilitate other functionalities crucial to lab assistance, such as picking up and placing objects from the cart the robot walks with.

Once code implementation was completed, testing to find Koalby's optimal walking motion was done in simulation using a model of the robot grasping the cart. The skeleton of Koalby we created for unassisted stabilization testing was used in tandem with a cart created by attaching four wheels to a large rectangular object. To simulate Koalby grabbing the cart, the robot's right hand was attached to the cart with a joint, however the left hand was not able to be connected to the cart due to CoppeliaSim's limb hierarchy system not allowing that. This led to some balancing issues during testing, but the setup was stable enough to allow for ample testing of the walk cycle.

Using this simulated model, we mainly tested two different parameters to optimize Koalby's walk cycle. The first was the delay between the calculation of each point on the trajectory mentioned earlier in this chapter. The higher the value was, the fewer points that were calculated and vice-versa. This parameter affected the smoothness of the walking motion, with

less points leading to a more jagged trajectory. However, too many points would lead to a large computation time and would also result in jagged movements, so our final value for it was 0.5 seconds.

The second parameter was changing the angular position values of the three joints being controlled in the leg: the thigh extender, the knee, and the ankle. This was done easily by accessing a CSV file with the initial and final positions in it, and the angles for each joint in both positions could be edited for testing. We tested numerous different stride lengths and step heights by changing these values, analyzing how Koalby reacted to each step. Through testing, we found that steps that were too high would lead to the robot quickly losing its balance and falling over as all of its weight would be on one leg for too long and it would not be able to readjust by planting its foot back on the ground fast enough. In a similar vein, stride lengths that were too large would also lead to Koalby losing balance and falling down as the leg would either smack the cart in front of the robot or would not be able to lift itself up back into a standing position.

Through our testing, we found that the most stable step that prevented Koalby from falling over was one with a very short stride length and step height that mirrors how other bipedal robots in the industry walk. These values led to Koalby lasting the longest amount of time without falling over, being able to hold itself up for around 10-20 seconds while walking. A three-frame representation of how the step looked and the joint angle values that correspond with each part of the step that we decided on are shown in Figure 8.3. However, these angles were not perfect due to the aforementioned balance issues that came with not being able to attach both of Koalby's arms to the simulated cart, but using the simulation results as a proof of concept provided our team with enough information and confidence to test how Koalby walks in the real-world.

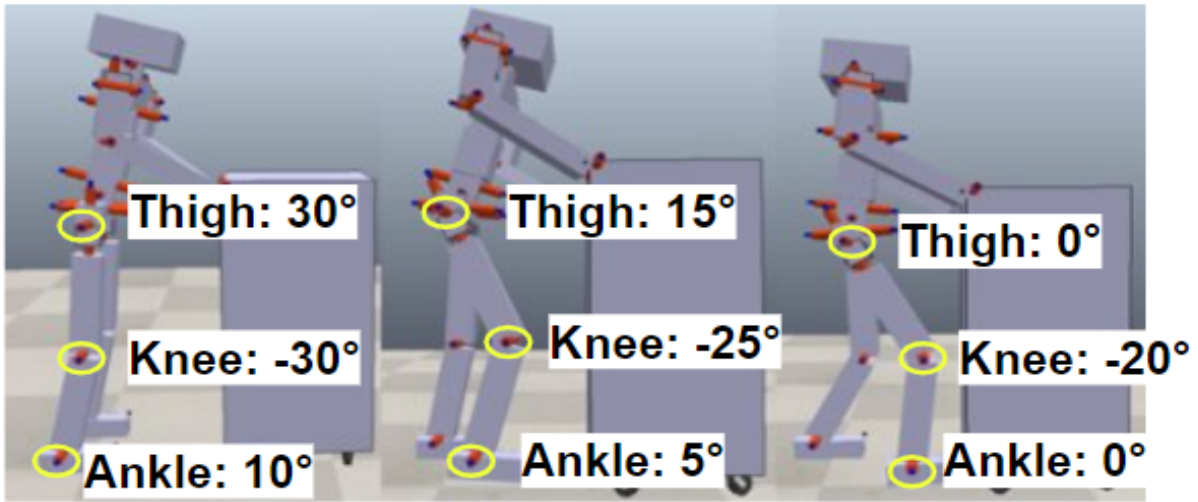


Figure 8.3: Progression of Koalby's Most Stable Step With Corresponding Joint Angle Values for Each Stage

With the software in place allowing for Koalby to walk, we ran tests on the real-world robot to find out if any modifications needed to be made. On the real-world robot, the steps taken through the use of trajectory planning had very jagged movements that were not suitable for walking when compared to its smooth simulation counterpart as there were some timing issues that needed to be debugged. However, we unfortunately did not have to test this further outside of simulation as we wanted to prioritize getting Koalby to stand stably assisted by the cart.

9. Electronics and Control

9.1 Zeroing Motors

In order to simplify the robot’s code, as well as make it easier to move between the physical machine and its representation in simulation, the team shifted the default positions of the motors. Based on documentation, it appears that the previous team simply installed the motors, moved them to a desired location, and read that position’s value through the Arduino code. This meant that every motor’s default position, while functional in the physical world, was represented by an entirely arbitrary value in code. To resolve this, the team went through a process of “zeroing” each motor — that is, disconnecting it from the robot, setting its default angle to zero, and then reattaching it so that its zero position matched where we wanted the connected limb to initialize. A sample of the software changes is shown in Figure 9.1. This resulted in cleaner documentation for both our team and future contributors.

			Motor ID	Max Angle	Min Angle	Initial Angle		
Initial	Motor RAnkle =	{0x00,	112,	45,	66,	HERK};	//20	* HerkLuex, right ankle - unplugged ???
Zeroed	Motor RAnkle =	{0x00,	46,	-21,	0,	HERK};	//20	* HerkLuex, right ankle - unplugged ???
Initial	Motor RKnee =	{0x14,	43,	-11,	-9,	HERK};	//19	* HerkLuex, right knee
Zeroed	Motor RKnee =	{0x14,	52,	-2,	0,	HERK};	//19	* HerkLuex, right knee
Initial	Motor LHipZ =	{0x0E,	25,	-5,	8,	HERK};	//12	* HerkLuex, right hip 2 - NEEDS ADJUSTMENT
Zeroed	Motor LHipZ =	{0x0E,	17,	-13,	0,	HERK};	//12	* HerkLuex, right hip 2 - NEEDS ADJUSTMENT
Initial	Motor RHipX =	{0x08,	17.55,	-14,	6,	HERK};	//16	* HerkLuex, right hip 1
Zeroed	Motor RHipX =	{0x08,	12,	-20,	0,	HERK};	//16	* HerkLuex, right hip 1
Initial	Motor LHipX =	{0x09,	0,	16,	13,	HERK};	//11	* HerkLuex, right hip 1
Zeroed	Motor LHipX =	{0x09,	3,	-13,	0,	HERK};	//11	* HerkLuex, right hip 1
Initial	Motor Abdomen =	{0x13,	-166,	0,	-95,	HERK};	//10	* Motor 13 - HerkLuex, Abdomen
Zeroed	Motor Abdomen =	{0x13,	95,	-71,	0,	HERK};	//10	* Motor 13 - HerkLuex, Abdomen
Initial	Motor LAnkle =	{0x05,	-84.17,	-146.58,	-126.92,	HERK};	//15	* HerkLuex, right ankle
Zeroed	Motor LAnkle =	{0x05,	43,	20,	0,	HERK};	//15	* HerkLuex, right ankle
Initial	Motor LKnee =	{0x0C,	-63,	22,	19,	HERK};	//14	* HerkLuex, right knee
Zeroed	Motor LKnee =	{0x0C,	-82,	3,	0,	HERK};	//14	* HerkLuex, right knee
Initial	Motor RightForearm =	{0x01,	-1,	-125,	-7,	HERK};	//0	* Motor 1 - Herkulex, Right Forearm
Zeroed	Motor RightForearm =	{0x01,	6,	-118,	0,	HERK};	//0	* Motor 1 - Herkulex, Right Forearm
Initial	Motor RightArmZ =	{0x02,	-85,	102,	5,	HERK};	//1	* Motor 2 - Herkulex, Right Upper Arm
Zeroed	Motor RightArmZ =	{0x02,	97,	-90,	0,	HERK};	//1	* Motor 2 - Herkulex, Right Upper Arm
Initial	Motor RightShoulderX =	{0x03,	-20,	130,	-3,	HERK};	//2	* Motor 3 - Herkulex, Right Arm Connector
Zeroed	Motor RightShoulderX =	{0x03,	133,	-17,	0,	HERK};	//2	* Motor 3 - Herkulex, Right Arm Connector
Initial	Motor LeftForearm =	{0x0B,	-35,	99,	-32,	HERK};	//4	* Motor B - Herkulex, Left Forearm ???
Zeroed	Motor LeftForearm =	{0x0B,	131,	-3,	0,	HERK};	//4	* Motor B - Herkulex, Left Forearm ???
Initial	Motor LeftUpperConnector =	{0x06,	7,	-145,	-3,	HERK};	//6	* Motor 6 - Herkulex, Left Arm Connector
Zeroed	Motor LeftUpperConnector =	{0x06,	10,	-142,	0,	HERK};	//6	* Motor 6 - Herkulex, Left Arm Connector
Initial	Motor LeftShoulder =	{0x07,	95,	-83,	2,	HERK};	//7	* Motor F - Herkulex, Left Shoulder
Zeroed	Motor LeftShoulder =	{0x07,	93,	-85,	0,	HERK};	//7	* Motor F - Herkulex, Left Shoulder
Initial	Motor TorsoDRRear =	{0x11,	-150,	-30,	-94,	HERK};	//8	* Motor 11 - Herkulex, Torso Double Rotation Backside
Zeroed	Motor TorsoDRRear =	{0x11,	64,	-56,	0,	HERK};	//8	* Motor 11 - Herkulex, Torso Double Rotation Backside
Initial	Motor TorsoDRFront =	{0x12,	1.63,	-70,	-21,	HERK};	//9	* Motor 12 - Herkulex, Torso Double Rotation Frontside
Zeroed	Motor TorsoDRFront =	{0x12,	23,	-49,	0,	HERK};	//9	* Motor 12 - Herkulex, Torso Double Rotation Frontside

Figure 9.1: Changes in Motor Values From Zeroing

9.2 Dynamixel Versus HerkuleX Motors

Dynamixel motors generally have a more complex functionality, such as faster communication speeds and higher resolutions compared to HerkuleX motors, with the drawback of being more expensive. However, both Dynamixel and HerkuleX motors have similar functionality related to feedback and control. The HerkuleX DRS-0601 were compared to the Dynamixel MX-64AT as a possible replacement options.

The key differences between these Dynamixel MX-64AT and HerkuleX DRS-0601 motors are the resolution, operating angle, and communication speeds. The Dynamixel MX-64AT and HerkuleX DRS-0601 motors are shown in Figures 9.2 and 9.3. The Dynamixel MX-64AT has a more precise resolution of 0.088° and a larger operating angle of 360° compared to the HerkuleX DRS-0601 which is 0.163° and 320° , respectively. The Dynamixel MX-64AT has communication speeds of 8000bps ~ 4.5Mbps while the HerkuleX DRS-0601 is limited to a maximum of 1Mbps.



Figure 9.2: Dynamixel MX-64AT Motor



Figure 9.3: HerkuleX DRS-0601 Motor

Additionally, the HerkuleX DRS-0101 was compared to the Dynamixel AX-12 as a possible replacement for the neck motors. Figure 9.4 shows the HerkuleX DRS-0101 motor, and Figure 9.5 shows the Dynamixel AX-12. These motors have similar functionality differences as the Dynamixel MX-64AT and HerkuleX DRS-0601. The Dynamixel AX-12 has a more precise resolution of 0.29° compared to the HerkuleX DRS-0101 which is 0.325° . The Dynamixel AX-12 has communication speeds of 7343bps ~ 1Mbps while the HerkuleX DRS-0101 is limited to a maximum of 0.67 Mbps. However, the Dynamixel AX-12 has a smaller operating angle of 300° compared to the 320° operating angle of the HerkuleX DRS-0101.



Figure 9.4: HerkuleX DRS-0101 Motor



Figure 9.5: Dynamixel AX-12 Motor

For the purpose of this project, the resolution, operating angle, and communication speed of the HerkuleX motor was determined to be sufficient. Overall, the simple movements and gripping actions that were expected of this project did not require extremely precise or fast maneuverability. It was determined that the HerkuleX DRS-0601 and DRS-0101 are sufficient to replace the Dynamixel MX-64AT and AX-12 motors, respectively. However, the HerkuleX motor replacements are different dimensions than their Dynamixel counterparts, so this required redesigns in the parts to fit the new motors which were completed by the 2023 3D Printed Humanoid Robot Design MQP Team. Based on these replacements, Koalby now has 19 HerkuleX DRS-0201 motors, eight HerkuleX 0601 motors, and two HerkuleX DRS-0101 motors.

9.3 Sensors

As a result of performing literary research (Chapter 4.4) into the types of sensors used to establish more autonomy and stabilization in humanoid robots, three sensors were chosen: TF Luna for proximity sensing, BNO055 IMU, and the Huskylens AI Camera. The new circuit diagram with the new additions can be seen in Appendix E. The overall sensor integration can be seen and read more in Chapter 12.2 and in Figure 12.3.

9.3.1 LiDAR (TF Luna)

A LiDAR TF Luna sensor was chosen for proximity detection. This would be attached to the head of the robot to detect objects while walking to be able to stop or turn before hitting the obstacle. This sensor was first set up and tested individually before being integrated into the robot design.

Two different code bases were developed to run the TF Luna sensor on Arduino Uno and Arduino Mega boards. The Arduino Uno could use the arduino library SoftwareSerial which is automatically installed with Arduino. However, this library does not work well with the Arduino Mega, so this second code base uses an additional library, TFMPlus, that needed to be installed. Koalby required an Arduino Mega board which is why this second, more complex, code was developed to run the TF Luna on the Arduino Mega. Both code sources displayed the distance measured by the TF luna, the flux or strength, and temperature of the sensor in degrees celsius.

After the code was developed, the range of the TF Luna sensor needed to be tested. A basic testing setup was created to test the range, as shown in Figure 9.6. For this setup, the TF Luna sensor was taped to a box while a second box was placed in front of it to measure the distance between them. A tape measure was used to measure the distance and compare it to the distance measured by the sensor. The range was tested by taking multiple measurements from the sensor and plotting it against the actual distance, as shown in Figure 9.7. This sensor is advertised to have a range of up to 8 m, but the testing showed the ideal range to be up to 3 m. If measuring further than 3 m, the sensor reads nothing. Figure 9.7 shows that the TF Luna sensor is most accurate in the range of zero to three meters with an inaccuracy of one to 3 cm.

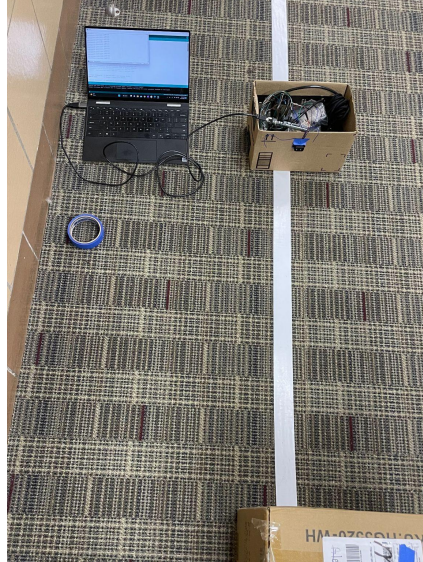


Figure 9.6: TF Luna Sensor Testing Setup

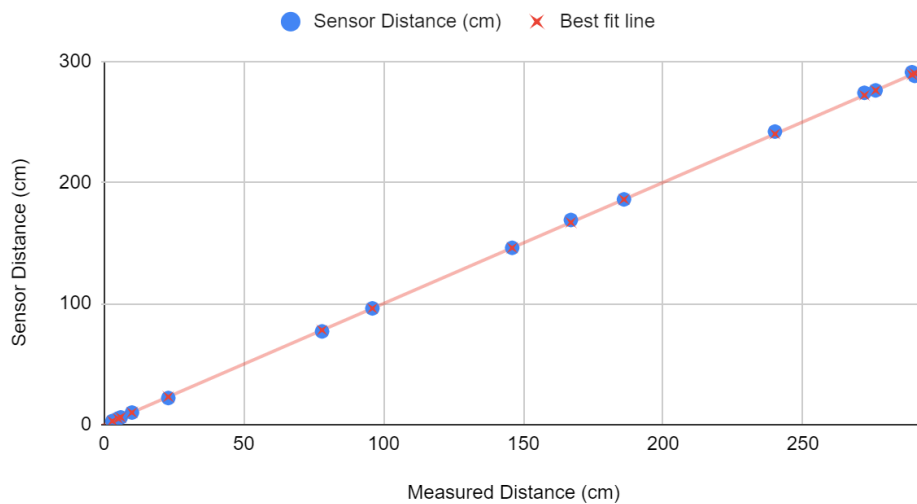


Figure 9.7: TF Luna Sensor Testing Graph

This testing also showed two important insights to keep in mind when integrating the sensor into Koalby: the angle of the sensor and how it is attached. If the sensor is slightly angled downward, then the sensor may read the distance to the floor before the intended object, providing incorrect data. This testing setup did not securely attach the sensor to the box to maintain a horizontal measuring angle which resulted in some incorrect data readings that needed to be re-measured. Therefore, we recommended that the attachment for the sensor is designed to hold the sensor securely horizontal for clean measurements. Additionally, loose wire

attachments during testing resulted in wires falling out, causing loss of function and making the testing inconsistent and more time-consuming. Thus, we recommended that the wires be securely attached or soldered when integrating the sensor onto the robot. Our partnered team, the 2023 3D Printed Humanoid Robot Design MQP Team, used these recommendations during their redesign of Koalby to integrate the TF Luna sensor into the robot.

9.3.2 IMU (BNO055)

An Adafruit BNO055 IMU was chosen for stability detection. The BNO055 is a 9-DOF IMU which performs its own sensor fusion [52]. This sensor was attached to the lower part of the torso and close to the center of mass which can be seen in Figure 9.8 below.

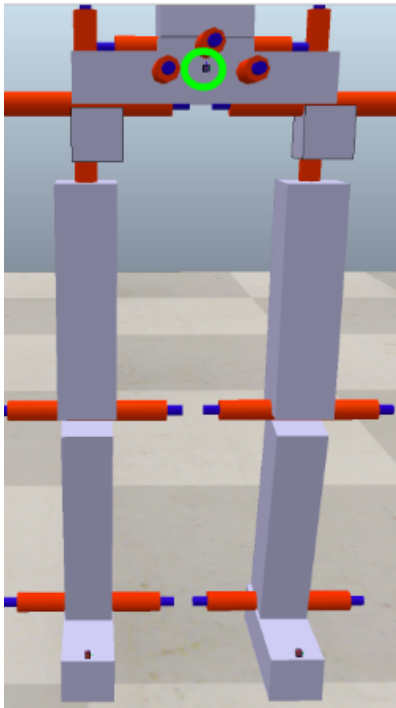


Figure 9.8: IMU Placed on CoM

This device was used to read the acceleration, gyroscope, and magnetometer values, which were then converted into roll and pitch angles to use with a PI controller to keep the upper part of the humanoid body stable. The sensor was first tested on the Arduino IDE for various movements and speeds. The IMU was tested on an Arduino Mega board and included the `Adafruit_BNO055.h` and `Adafruit_sensor.h` libraries.

The first test that was done on the IMU was leaving it at rest. The IMU was placed on the counter (Figure 9.9) and collected raw IMU readings for roughly 20 seconds at each test to find the X,Y, and Z acceleration relative to time.

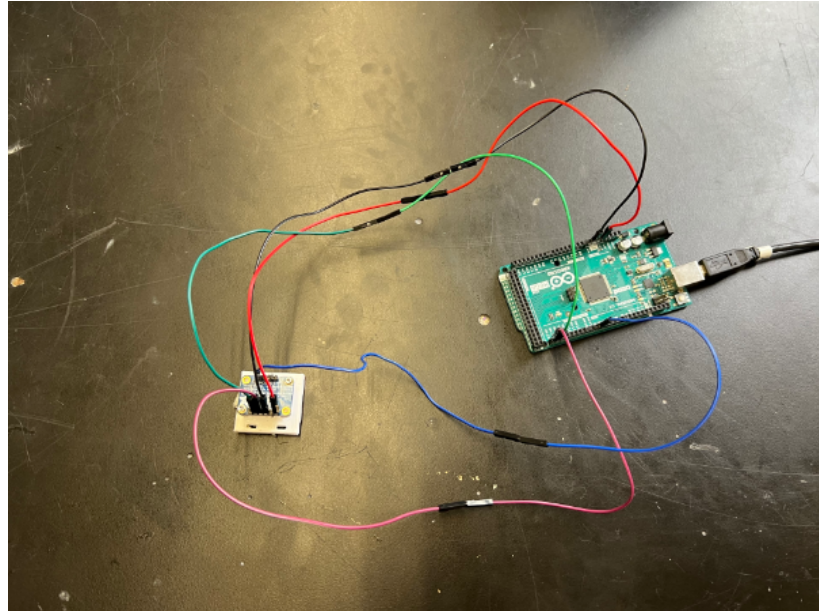


Figure: 9.9: BNO055 IMU Testing Setup

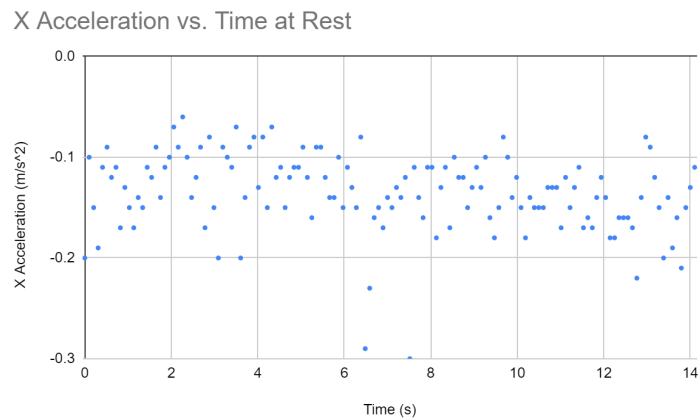


Figure 9.10: Acceleration of the IMU at rest in the X direction

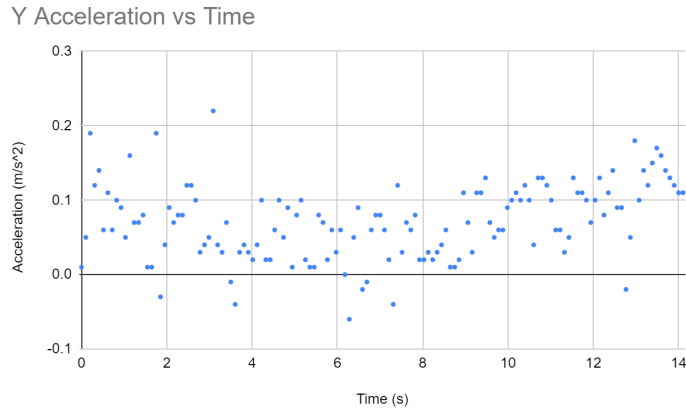


Figure 9.11: Acceleration of the IMU at rest in the Y direction

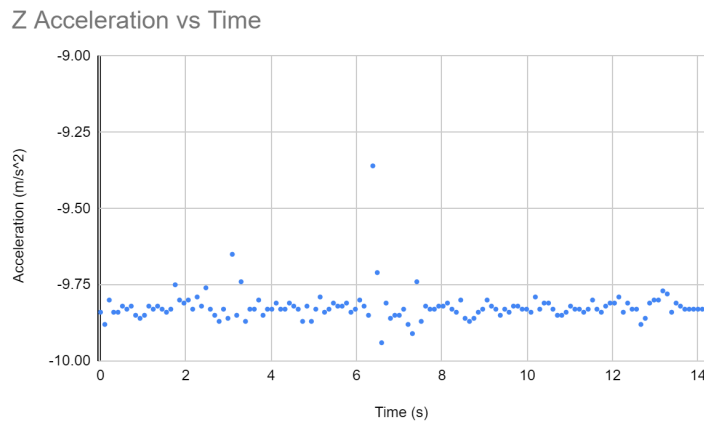


Figure 9.12: Acceleration of the IMU at rest in the Z direction

As seen in the above figures, 9.10-9.12, the X and Y accelerations were close to 0.2m² despite being at rest. This is due to the noise experienced by the IMU. The Z acceleration was around -9.8m/s² which was expected as gravity was acting on the IMU while at rest. Other tests that were performed on the IMU included moving the IMU along the counter slowly in one direction and then repeating the test at increased speeds. The graphs can be seen below:

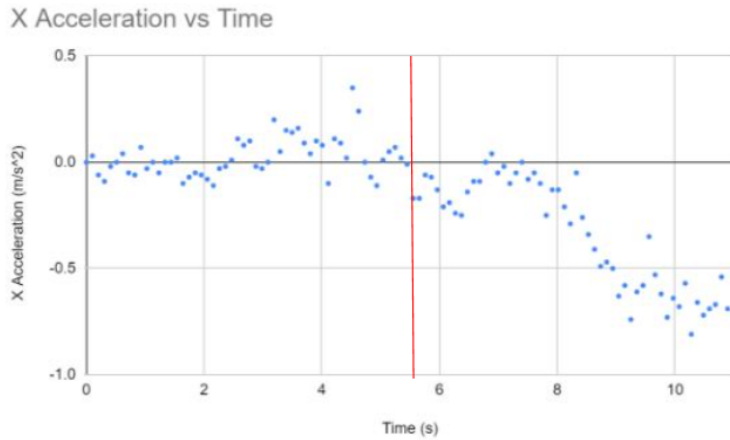


Figure 9.13: Acceleration of the IMU moving in the left direction slowly

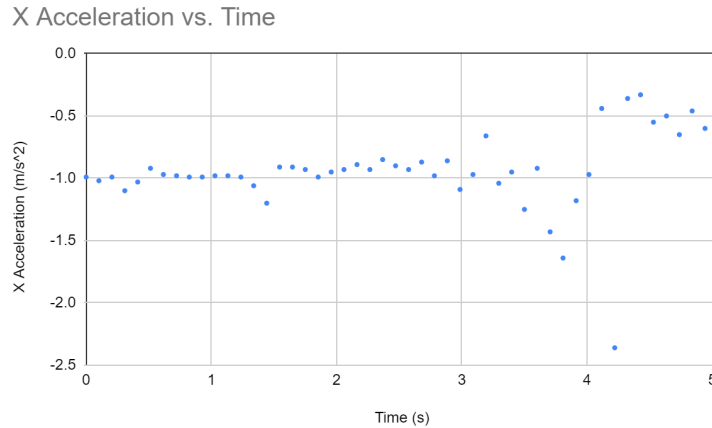


Figure 9.14: Acceleration of the IMU moving in the left direction fast

In these above figures, 9.13 and 9.14, it can be noticed that the IMU readings were negative when moving, due to the left axes being negative. In Figure 9.3.10, the IMU was very slow to move from zero until the red line just under 6 s then the acceleration started becoming negative and it finally got to around -0.7m/s^2 at the 10 s mark. In Figure 9.14, the IMU was moving left up until the 4 s mark. At that time the IMU started moving right until becoming stationary. This shows that in the X direction, the left is negative while the right is positive. By performing these tests we were able to establish a coordinate system for the IMU which can be seen below in Figure 9.15 and we were able to determine that the IMU does pick up noise.

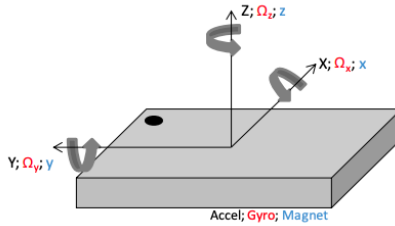


Figure 9.15: IMU Coordinate system

9.3.3 AI Camera (Huskylens)

An AI Camera, Huskylens, was chosen for proximity and object detection. The camera was tested by holding the camera 30 cm above, 25 cm away, and tilted 22.5° down towards the paper. This is simulated based on where the Huskylens would be mounted to the top of Koalby's head to detect objects. The camera was initially tested on an Arduino Mega board, using the Huskylens library. There were three main areas of testing the Huskylens: color detection, object recognition, and line tracking. The code can be seen in Appendix D.

The Huskylens had a mode which allowed the camera to learn and recognize colors. To train it, a blank piece of paper with one bright tape was used. Once this was achieved, the next step was testing whether the camera could identify the same color when mixed in with other unknown colors. The camera was successfully able to do so, and from there the next step was to train a second color and be able to differentiate the two. In order to train the Huskylens, the colored tape would be placed in the camera's frame of view for roughly 5 seconds which allowed it to pick the pixelated points associated with that color. This was done using one piece each of two different colorful tapes on a blank piece of paper, pink and yellow. Once the camera had been trained, the two tape colors were arranged on the paper for the camera (Figure 9.16) to identify which can be seen in Figure 9.17.

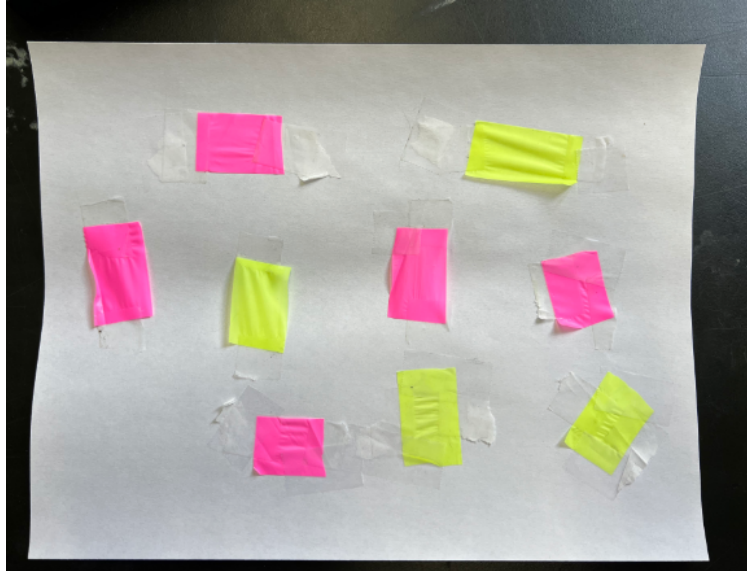


Figure 9.16: Paper with Color Tape for Training



Figure 9.17: Huskylens trained on two colors

The next set of training was object detection. In order to create a greater sense of autonomy in Koalby, identifying common laboratory objects (hammer, screwdriver, nail etc.) is crucial. The Huskylens camera comes with pre-programmed objects that the camera is able to recognize as seen in Figure 9.18. These pre-programmed objects are common everyday objects such as a bottle, car, and bird that was trained on the Huskylens using multiple images.

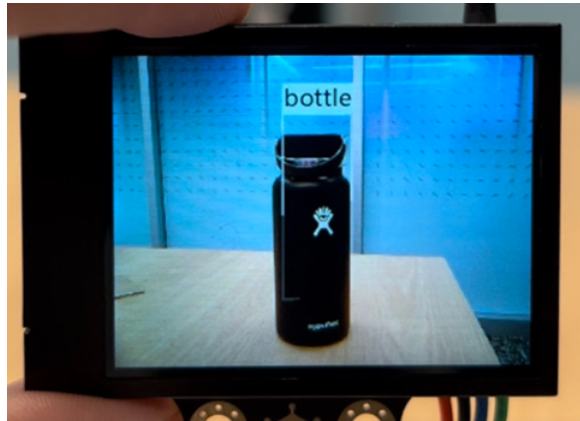


Figure 9.18: Correct Classification of Object

There were no preprogrammed objects that fit the scope of laboratory assistant so manual training was conducted by using multiple pictures for an object. More common lab objects (screwdriver, hammer, pliers) were attempted but the camera would not recognize the objects as unknown and it would just assign an incorrect preprogrammed classification.

The final testing of the Huskylens was line tracking. This function was expected to be helpful as it could potentially allow Koalby to walk in a set path within a laboratory setting, however the camera was very noisy and the arrow marking the line would jump around and not give a precise reading. In the line tracking mode, the blue arrow is where the camera thinks the line is and it prints the x and y of the arrow origin and the x and y of the target position (the end of the line). This can be seen in Figure 9.19 taken within 2 s of each other.

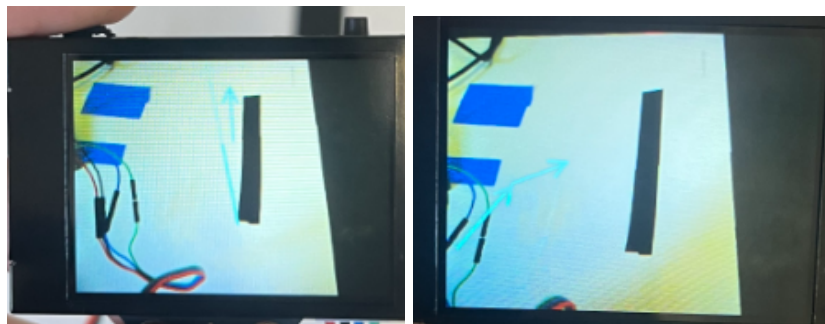


Figure 9.19: Blue Arrow Attempting to Follow Black Line

Due to the noise of the line tracking, the Huskylens was only used for color and object detection. After testing, it was very apparent that the Huskylens is very dependent on lighting, which is common for any camera-based system, so for optimal results, the training conditions should be similar to the testing lighting. The lighting used was in the MQP laboratory on a sunny day in the afternoon. The Huskylens were also found to not be as accurate at identifying colors when it was more than 40 cm away.

9.4 Electrical Integration

Due to the addition of sensors and new motors, a new electrical diagram was constructed for the robot (see Appendix E). To accommodate for the different voltages across the sensors and motors, a series of adjustable voltage regulators were implemented. Meanwhile, the previous 7.4V batteries were upgraded to two 11.1V batteries to provide sufficient power. Then to improve the durability and reliability of the circuit, old and worn components were replaced and higher quality, threaded wires replaced the solid core wires.

10. Simulation Software

While Blender is an extremely versatile physics simulation, it is also quite demanding on machines that run it. In addition, Gazebo and CoppeliaSim were both developed specifically with the intention of simulating robots, so Blender lacked much of the streamlining and special features possessed by the other two softwares, particularly the specialized sensors being incorporated into the robot. Furthermore, Gazebo, while providing extensive support and plug-ins that could potentially improve workflow, lacked compatibility with Windows, the primary operating system of our team. CoppeliaSim, therefore, was left as the best candidate for our purposes. CoppeliaSim, developed as a successor to the V-REP software discussed in the background chapter, is an exceptionally robust and customizable software for simulating robotic systems.

Within the CoppeliaSim environment, different versions of the software offered different functionalities. The two primary points of interest for our team were the ability to implement Python scripts directly into the simulation and methods of joint control. Within CoppeliaSim, robots are constructed using joints and links. In order to determine the positions of these links and joints, the team measured the motor-to-motor distance throughout the entire robot. The joints were then placed using those distances, and connected using joints.

In CoppeliaSim, movement had to be programmed into the simulated robot using either the simulation's custom programming language or by connecting an externally-running Python script to an online server, which was then subsequently connected to the running simulation. Both of these methods of movement control are slow and require additional code that must be discarded when working with the physical robot. Because the two major features that were important to our team's work, torque-force mode and embedded Python, were never available on the same version of CoppeliaSim, our team had to choose one of the two to prioritize. Ultimately, we determined that, while the additional steps of indirectly running Python code in older versions was unideal, it was most important to accurately reproduce the construction and functionality of the robot and its joints. Therefore, our team performed all of our robot simulations in CoppeliaSim version 4.2 to still allow the use of torque-force mode.

CoppeliaSim also has many built-in features to ease model creation and simulation, like geometric shapes, placeable and configurable joints, and simulated sensors. These features made

testing of the robot's existing codebase in simulation easier, quicker, and more accurate in comparison to other simulation environments. For example, in order to test the robot's stabilization, which relied on recognizing input from an IMU placed on the robot and adjusting motor positions based on the IMU data in the real world, the simulated robot could be assigned a combination gyroscope and accelerometer to replicate the functionality of an IMU. The outputs of these sensors could then be read by the robot's control code in the exact same way as the IMU is read in real life, and thus its ability to keep the robot stable acts as a truly reliable indicator of the feasibility of the IMU successfully stabilizing the real robot. These same principles applied to the proximity and visual sensors that were also incorporated into Koalby.

10.1 Building Koalby in Simulation

At the start of our efforts to build Koalby in our simulation environment, we first tried using URDF files to import the robot's model and its kinematics into CoppeliaSim. We tried two different files, one that was left over by the 2021-2022 MQP team's efforts on the project and another that we generated from a 3D computerized model of the robot. The one used by the previous team gave many errors upon importing it into the software that our team was unsure how to fix and the one our team generated had many confusing errors like importing pieces of Koalby upside-down and other oddities. Our attempts to fix these URDF files proved fruitless, so we decided that our best alternative was to recreate Koalby as a skeleton using CoppeliaSim's ability to manually import shapes and joints into its simulation environment. The robot's limbs were made up of rectangular prisms connected by joints, and the limbs themselves were modeled in accordance with the actual measurements of the real-world robot. The constructed model can be seen in Figure 10.1.

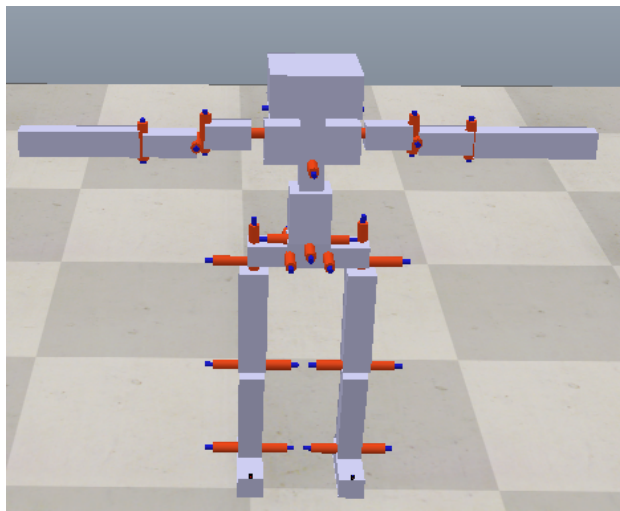


Figure 10.1: Initial Koalby Model in CoppeliaSim

While some of Koalby's motors are attached to the same part in the real world, this same idea could not be replicated in CoppeliaSim. Each joint had to be attached to its own rectangular prism in order to properly simulate the movement functionality. If two joints did not have a prism between each other, the one of the joints had no effect when it was moved.

11. Stabilization

11.1 Determine Position and Motor Angles

As Chapter 4.7 states, the most stable position of a person is where their knees are slightly bent, lowering the center of gravity relative to the ground. The feet must also be shoulder width apart, creating the most stable base for standing. In order to prevent tipping of the upper half, arms should be stretched out to the side. With this research in mind, Koalby was set to this position in the simulation in order to find his most stable standing position. His feet were manually positioned to be 0.19m apart, the distance of shoulder width. Then, each of his joints were set to 0° , then physically rotated to what was a T-pose. We tested this initial stance to get a baseline, and Koalby fell backward. From there, his knees were each bent to -20° , so that his CoM would be closer to the ground. His ankles also needed to be moved, in order for his feet to be flat on the ground. His ankles were set to 20° , and the simulation was run again. Koalby fell

forward very quickly, so his knees were set to be slightly less bent, at -10° . His ankles were also adjusted, correspondingly, to 10° . At this point, he fell forward slower, but still was not stable.

From there, we tested a staggered stance. His left leg was positioned slightly in front of his body, with the hip at 10° , the knee at -20° , and the ankle at 30° . His right leg was positioned slightly behind his body, with the hip at -10° , the knee at -20° , and the ankle at 30° . This stance resulted in Koalby falling diagonally forward and to the right. Other tests only produced a fall in one direction, so this stance was aborted due to the increased difficulty in finding a balancing point.

We reset to having his feet even with each other, and decided to adjust the bend in his hips. He was falling forwards, so his front to back movement joint was set to 10° . From there, the front to back joint was tested with trial and error. We first adjusted the angle by 1° , and found that between 5° and 6° Koalby was stable for the longest. After this discovery, we incremented by $.1^\circ$, and found the most stable front to back angle was 5.2° . This trial and error process led to the discovery of his most stable position overall, pictured in Figure 11.1 This was before any sensor was attached or integrated into Koalby so we could have a control reading.

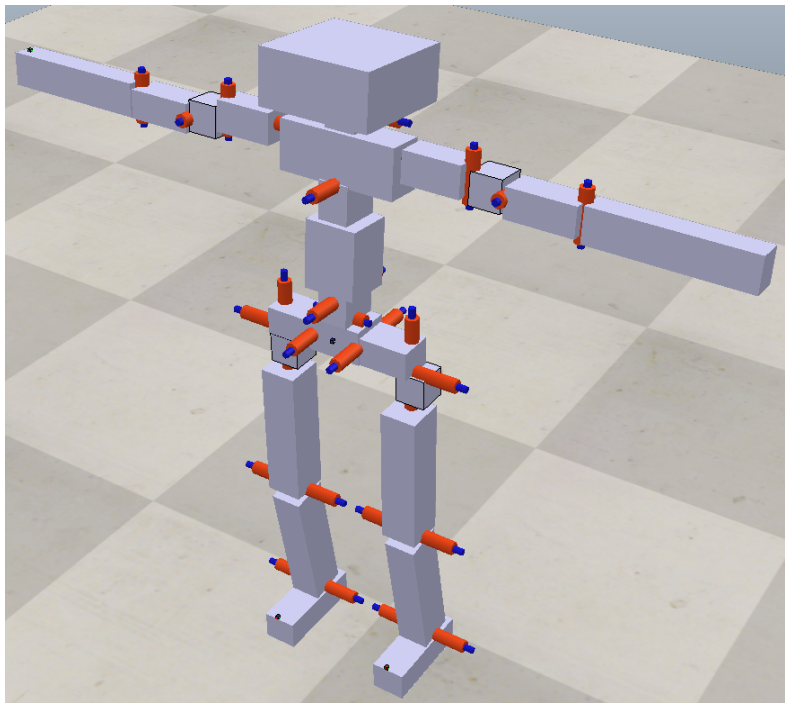


Figure 11.1: Koalby's Most Stable Position

After this position was discovered, with the knees bent to -10° , the ankles at 10° , and the front to back joint in the hips at 5.2° , Koalby's arms were adjusted to reflect holding a cart. His elbows were set to 70° , his shoulders were set to 90° down and rotated 20° inwards. From there, a cart was added to the simulation. In order to get his hand attached to the cart, we attached a static joint in-between his right forearm part and the cart. Without this joint, Koalby's hand did not connect to the cart in any way. The trial and error process was repeated to find the most stable position with the cart. Again, his most stable position was discovered. This position mimicked the stable position without the cart, aside from the arms, with the only exception being that the front to back joint was -10° . This allowed for Koalby to be leaning forward, making the CoM of himself and the cart closer to the middle of the system. This stable position with the cart can be seen in Figure 11.2.

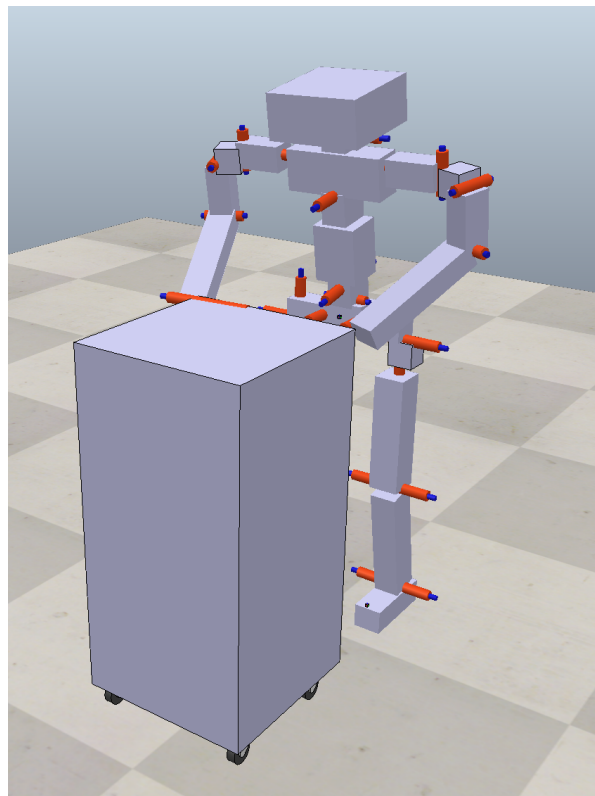


Figure 11.2: Koalby standing in simulation with a cart

11.2 Gathering IMU Readings and Stabilized Controls

Gathering IMU readings and implementation of a PI control feedback loop were primarily done in simulation. CopelliaSim has a sensor library which allows for a drag and drop

feature. This was used to place a gyroscope and accelerometer in the simulation. The two sensors were placed at the center of mass (approximately 0.583 m from the ground) which can be seen in Figure 11.3:

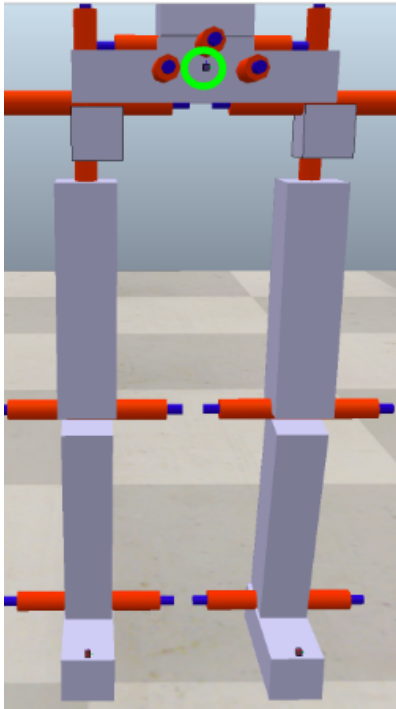


Figure 11.3: Gyroscope and Accelerometer Placed at Center of Mass

CoppeliaSim had its own PID controller, which was used to test stabilization. The IMU readings were found at the most stable standing position in the simulation. The values found were 67.5° for yaw, -1.9° for pitch, and -0.2° for roll. In order to find the exact PID contents, values were estimated until the values kept Koalby stable. Through trial and error, the proportional, integral, and derivative constants were found to be 50, 2, and 0 respectively. These parameters worked to keep the simulated robot standing for more than five minutes with PI control in simulation as seen in Figure 11.4.

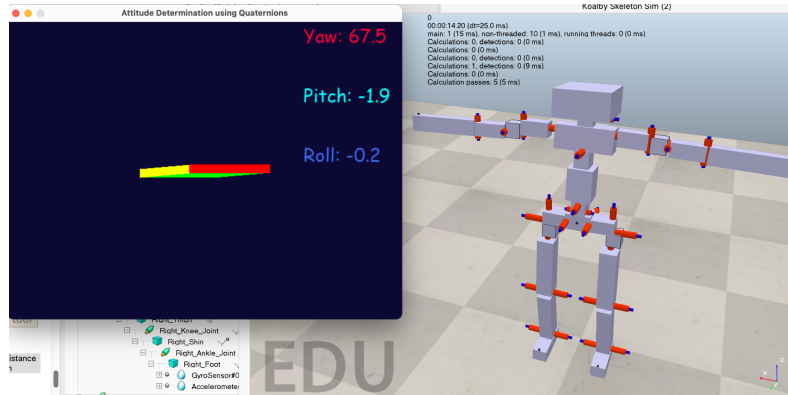


Figure 11.4: Koalby stable in simulation with PI control

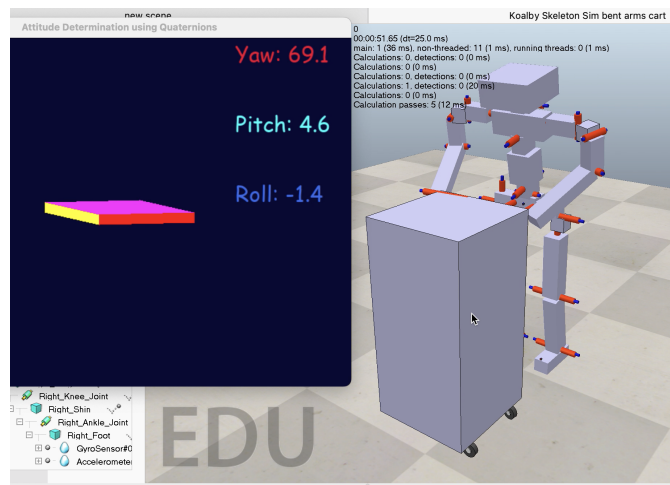


Figure 11.5: Koalby Standing with cart

In Figure 11.5, Koalby is seen to be standing assisted with the cart and the IMU at the center of mass is reading the IMU values in real-time. With IMU values of 69.1° for roll, 4.6° for pitch, and -1.4° for roll. This proof of concept allowed stabilization testing in the real world attainable. These values will not be identical to the real world but they are a start to achieving assisted standing using the IMU and PI control.

11.3 Choosing a Filter

As mentioned in Chapter 4.5, IMUs are notoriously noisy, and in order to combat this we decided on using the Kalman filter. Despite the Kalman filter being more complex compared to the complementary filter, the predictions are more accurate. In addition, the Kalman filter does not store a large amount of memory and quickly loops through the predict, measure, and update states as mentioned in the Background. Quick and accurate processing of IMU readings were required in order to keep Koalby stabilized. The Kalman filter was applied to the code and can be seen in Appendix D. The breakdown of the code can be seen in Figures 11.6-11.7 which show how the IMU collects raw data, then passes it through the Kalman Filter (predict and update steps) and then it goes through the PI controller which controls the motor positions and these cycles. The filter was for the IMU in the real-world and was not actually tested due to time constraints.

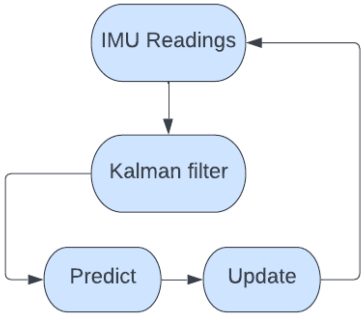


Figure 11.6: Flowchart of the IMU Readings with the Kalman Filter

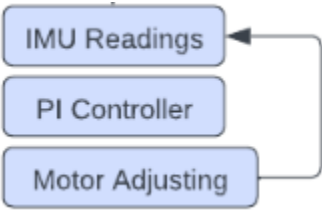


Figure 11.7: Flowchart of the IMU Readings with the PI Controller

11.4 Testing Stability

In simulation stability trials with the sensors integrated, it was determined that Koalby was most stable in the same position as before, but stood for a longer amount of time with the PI

control loop enabled. In an average of five trials, timing when the simulation was started to when Koalby's body hit the ground, it was found that Koalby stood for about 7.6 times longer with the PI control loop enabled. Refer to Table 11.1 to see the chart.

Table 11.1 Stability Trials With and Without PI Control

Stability Trials	
Without PI (s)	With PI (s)
2.21	18.98
2.28	16.39
2.43	18.41
2.28	15.94
2.11	16.39
Average	Average
2.262	17.222

From these trials in simulation, ideas were transferred to the real world and similar tests were conducted. In the real world, we replicated the same position from the simulation and tested that stance. The real robot fell slightly forward, so conducted the same trial and error process to find the best angle for the front to back joint, leaving the knees in place. After the process was finished, we found the most stable position for the front to back joint was between -7° and -12° . Unfortunately, it was difficult to be as precise in the real world, due to outside factors such as loose screws varying the most stable position.

We also had to adjust Koalby's arms, because in simulation, the cart is about .636m tall, and in the real world, the cart is .582m tall. The cart is also about 0.291m wide, and in real life, the cart is 0.424m wide. This meant his arms had to be out wider and reach down more. After these small adjustments were made, we had found his most stable position, shown in Figure 11.8. His hands are taped to the cart, and his feet are about shoulder width apart, 0.172 m, with his knees slightly bent.

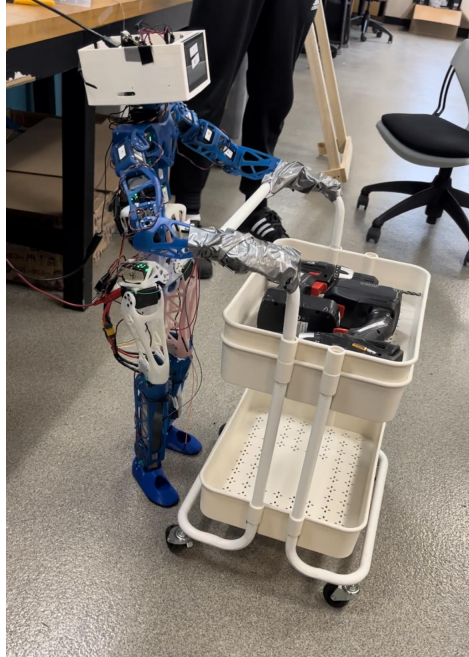


Figure 11.8: Koalby Standing in the Real World

From this stable position, tests were conducted to see if Koalby could perform basic movements while maintaining this balance. These basic movements included waving, and doing a handshake motion with his right hand. Only doing movements with his right hand allowed him to still have his left hand attached to the cart for support. It was determined that he could stay stable and standing with only one hand gently taped to the cart, while waving or shaking hands with the other. Figure 11.9 shows Koalby performing a handshaking movement with his right hand.



Figure 11.9: Koalby Standing with a Cart and Waving

Tests of unassisted standing were limited due to time constraints, however, the most successful test conducted was having him in the same position as the simulation, with his arms outstretched, and he was held from his pelvis by one finger. This test can be seen in Figure 11.10.

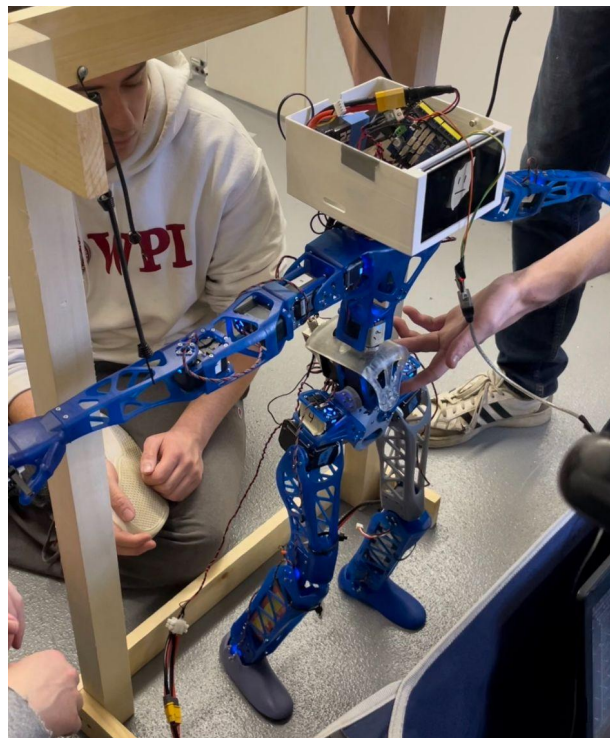


Figure 11.10: Koalby Standing Assisted by Only One Finger

12. Koalby's Software

12.1 Code Refactoring

In order to achieve all the desired functionalities with one code project, the Python code needed to be refactored.

12.1.1 Incorporating the Simulation

Getting Koalby working in the simulation was a main goal of the project, so it was addressed first. The existing Python code was able to be run via CoppeliaSim with a few tweaks. We needed to add a simulation folder, with remote API functions provided by CoppeliaSim. With these, we could connect to the simulation with 4 additional lines of code.

In order to physically connect, the simulation needed to be running, then the Python code could use the CoppeliaSim library to establish a connection. The motors' names also had to match in the simulation and the Python code so they could be controlled, so those were edited as well. From there, abstraction was done to reuse code from the real robot in the simulated robot.

12.1.2 Improving Usability and Readability

The previous code contained lots of threads that were unnecessary. Threads allow the code to be split up into smaller tasks and run concurrently, however, they make the code harder to debug and put stress on the operating system's processing time and memory. In the existing code, executing movements was threaded, recording movements was threaded and updating the robot was threaded. These processes were not handling large tasks, and ran at the same speed when they were not threaded. Additionally, when trying to debug these portions of the code, it was difficult to determine where the issues were with the threads. With this, we determined that the negatives outweighed the positives in this case, and removed the threads.

Additionally, with the removal of these threads, the code could be simplified and more widely re-used between different functionalities. For example, the pre-existing `ReplayPrimitive` class, used to play and record motions by looping through each motor and setting or getting its position, contained an object that contained information about the user-specified time it would take the robot to move from one position to the next, and even kept track of the motors, which

was already accounted for in the robot object. In the refactoring, this class was simplified into a MovementManager class. The new MovementManager class's only job was to play existing motions and record new motions. The rest of the information required to do this was passed from other places, making the code more readable and concise.

Lastly, the previous code's overall structure and naming conventions were not consistent or methodically done. The code did not use consistent naming conventions for variables and methods. In some places, CamelCase was used, and others used Python naming conventions. The team went through and changed all variables and methods to be lowercase and contain underscores between words, matching the Python naming conventions. The old code flow chart can be seen in background in Figure 2.5.1 and the new design can be seen in figure 12.2.3.

12.1.3 Testing

As for testing, similar methods were used from last year, in that there were different files that could be run to test various functions, such as recording a new motion or playing a previously existing motion. However, a new section was added to allow the user to specify whether or not they are running the simulation. The code works the same way, but if the simulation is being run, it will connect to CoppeliaSim instead of sending signals to the Arduino. There was also functionality added to test the reading of IMU data from the simulation and the real world.

12.2 Project Architecture

As for the actual project architecture, the UI prototype was created using HTML and CSS. These tools were helpful in creating the layout and style of the application. In order to make buttons functional, Javascript was used. For the backend, Python was already being used so the decision to remain in Python was easily made. The web application was created using Flask due to its Python-based infrastructure and accessibility. All of the code itself was written in JetBrains' PyCharm IDE because it is capable of handling all of these capabilities.

We make the connection from frontend to backend with a rest API. A rest API allows for the separation of the client implementation from the server implementation. Clients send requests and the server responds to the requests. These requests are sent through the website URL and received with GET and PUT methods. Responses will then send status codes which are

interpreted and rendered to the client. This implementation makes it so the backend can run on the Raspberry Pi and the frontend can be hosted via a computer. As seen in Figure 12.1, the Raspberry Pi hosts the interface that can be connected to wirelessly and sends information to the Arduino via USB serial communication.

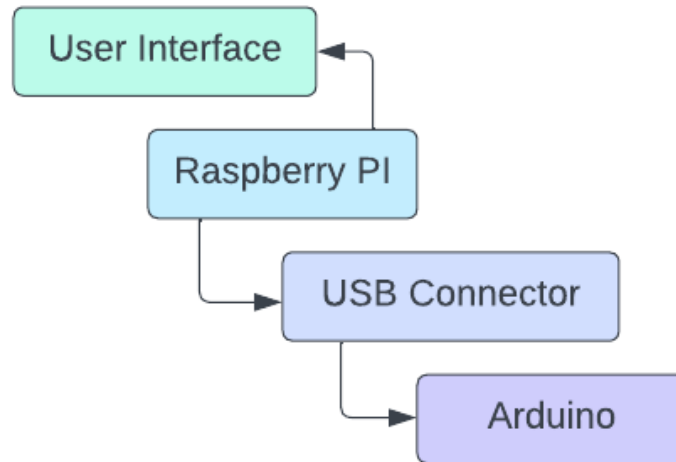


Figure 12.1: Code architecture

The user interface was successfully able to be hosted from the Raspberry Pi and connected to via a phone's hotspot, as seen in Figure 12.2. Users were able to initialize Koalby and make him execute pre-recorded movements by navigating to the pre-recorded movements page, selecting a movement to perform, and pressing run.

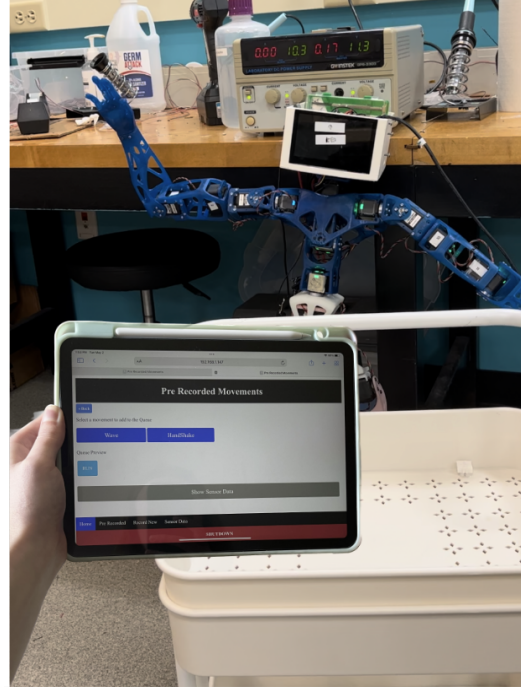


Figure 12.2: Koalby waving from the UI

One issue with this connection is the processing time. In initialization tests, where we timed how long it took the robot to initialize starting when we ran the “RunToInitializeKoalby” test file, it took an average of 6.66s, and the test table can be seen in Table 12.1. This time could be improved with faster serial communication. Additionally, in other places in the code required a communication delay. For example, when reading IMU data, if the Python code did not delay, IMU data would only be read every third loop iteration. However, when it waited for .25s, data was printed every loop iteration. Loops increase processing time in Python, so it is better to print data out every time, making the most of the iterations.

Table 12.1: Initialization Time Trials

Initialization Time (s)
6.7
7.3
6.2
6.9
6.2

The final code design is as seen in Figure 12.3. This shows how all code elements flow together, with the UI being the top most element, sending signals to initialize the robot. As mentioned in Chapter 12.1.1, this controlled either the simulated robot or the real robot. In this initialized state, the robot would stand stable, without the help of any sensor readings. From here, the robot was able to record a new movement (only in the real world), execute a preexisting movement, or stand stable with IMU readings. The stable stance required IMU readings, which are read, filtered, and passed to the PI controller, which adjusted the corresponding front to back and side to side motors. The robot also had the option to walk or reach for an object from this initialized state. To walk, it would use trajectory planning to calculate the trajectory of its desired step, move the motors corresponding to the trajectory, and repeat this process over and over for each leg. While this process was happening, it would read from the TF Luna to see if any obstacles were in the way, and stop walking if it got too close. On the other hand, if the user wanted Koalby to reach for an object, the robot would first see if the Huskylens detected an object for him to grab. If it did, he would use inverse kinematics and trajectory planning to figure out where the best trajectory to move his arm to reach the object.

This flowchart combines the Arduino and Python code, in that the sensor readings came directly from the Arduino, but were parsed and read by the Python code, allowing them to be processed and used in their respective ways. While not all of these components were implemented and tested on the real-world robot, the foundations for them are all present.

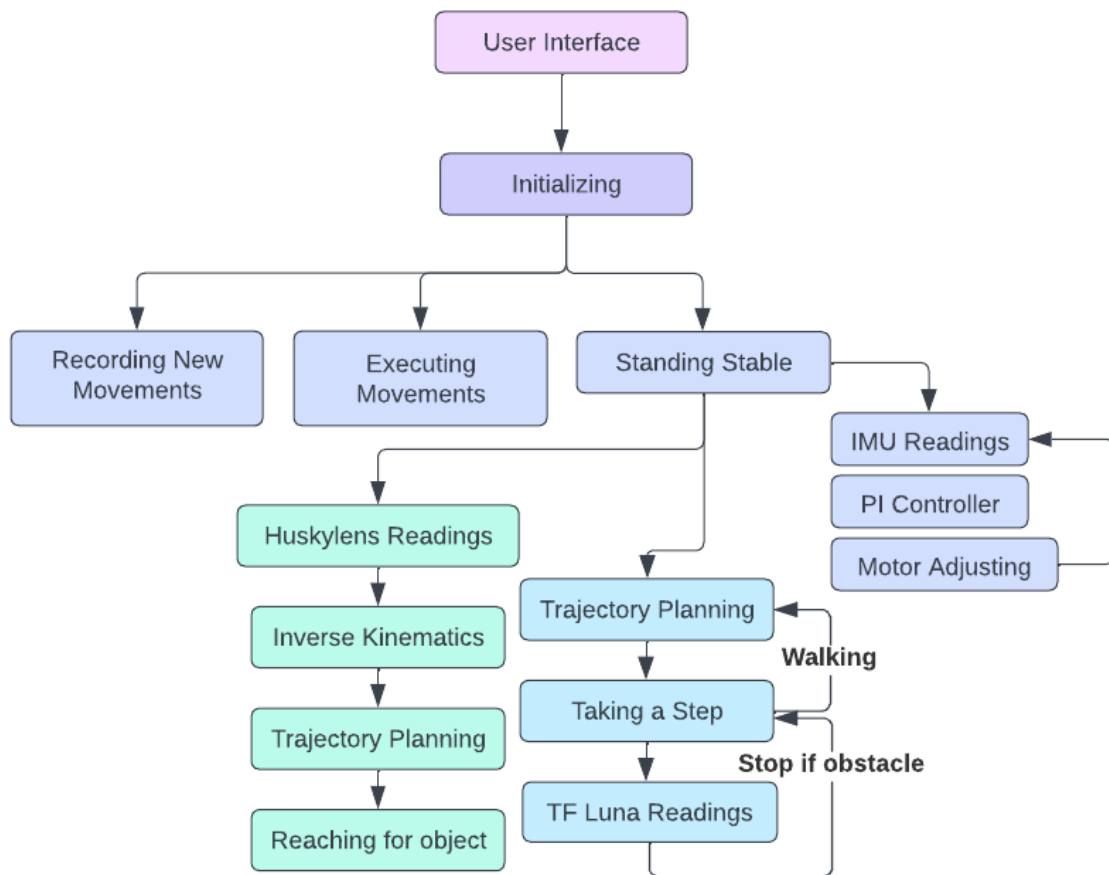


Figure 12.3: Overall code design

13. UI Prototyping

Due to the removal of threads and the need for the user to be able to connect to the robot wirelessly, the previous UI was discarded and a new one was developed from scratch. Different features and designs were discussed amongst the team based on robot functionality. We knew we needed to keep the design simplistic, while also providing full robot control. After this brainstorming session, each team member individually created a low-fidelity (lo-fi) prototype with the conversations in mind. We then combined all of our ideas into one and made another mockup.

13.1 Pre-Recorded Movements

It was important to have buttons for executing pre-recorded movements, as this was an existing functionality from last year. We also wanted to give the user control in deciding which movements were to be executed in what order, and for how long, so we added an editable queue preview to our design sketches. As seen in Figure 13.1, the pre-recorded movements page, the user had the ability to swipe through different movement options and add them to the queue. Once added to the queue, these movements would appear in the queue preview. If the user clicked next, they would be brought to the queue editing page in the right image. From here, the user could edit how long they wanted each movement to take and how many times they wanted it to repeat.



Figure 13.1: First UI LoFi Prototype Mockup of Pre-Recorded Movements (left) and Queue Editing Page (right)

13.2 Recording New Movements

Additionally, it was important for us to have a page to record new movements, as this was also an existing feature. We also wanted to give the user control in deciding how many positions they wanted to record, and the ability to play the recording while it was in progress of being created. Figure 13.2 shows the first mockup of this page, with a button to record the position of the robot, a field showing how many positions have already been recorded, a reset button to clear the recorded movements, a finish recording button to say the process was done, a play button to play the recording back, and finally a save button.

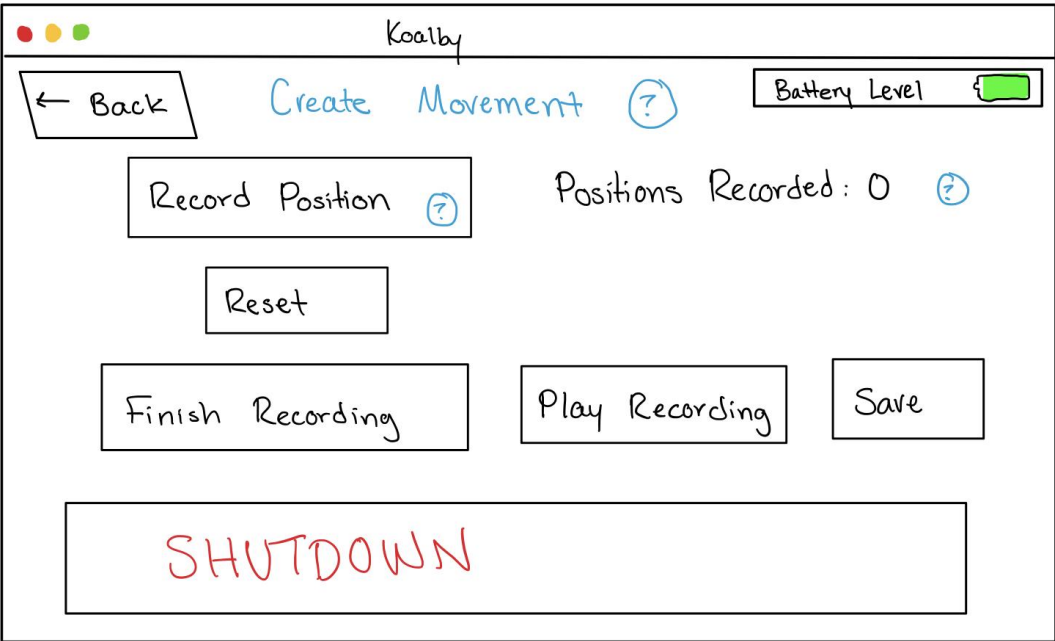


Figure 13.2: Creating Movement Page

13.3 Evaluating Prototypes

After this first iteration was completed, Nielsen’s Ten Heuristics [41] of a good application were discussed and measured within each person’s drawing. A heuristic evaluation was performed, as described in Chapter 4.8 with these key ideas in mind, and it became clear that different aspects needed to be fixed. One main aspect needing attention was the editable queue. It was too difficult for users to understand they could control how long each movement took, so

that was removed. The flow was also confusing because there was no defined way to get to the screen in which a user could control Koalby's walking. Additionally, user control was limited in the recording of new movements, because the user could not give a name to the movement they were creating. These issues were addressed and fixed.

After edits to the first iteration, informal feedback was collected from a variety of our peers. Some of these people were part of the 2023 3D Printed Humanoid Robot Design MQP Team, so they knew what we were trying to accomplish. Others were people who had limited knowledge of our project and the code itself. This informal feedback was minor suggestions, like including a gif of the robot's movement. It was also suggested to describe some movements, as some users may not be aware of what various movements are.

13.4 More Iterations

Edits were again made with the feedback received, and the mockup of a described movement is shown in Figure 13.3. The idea behind this was a user could click on the help button for a movement and a page having a gif showing the movement, and a description would pop up, describing what would happen.

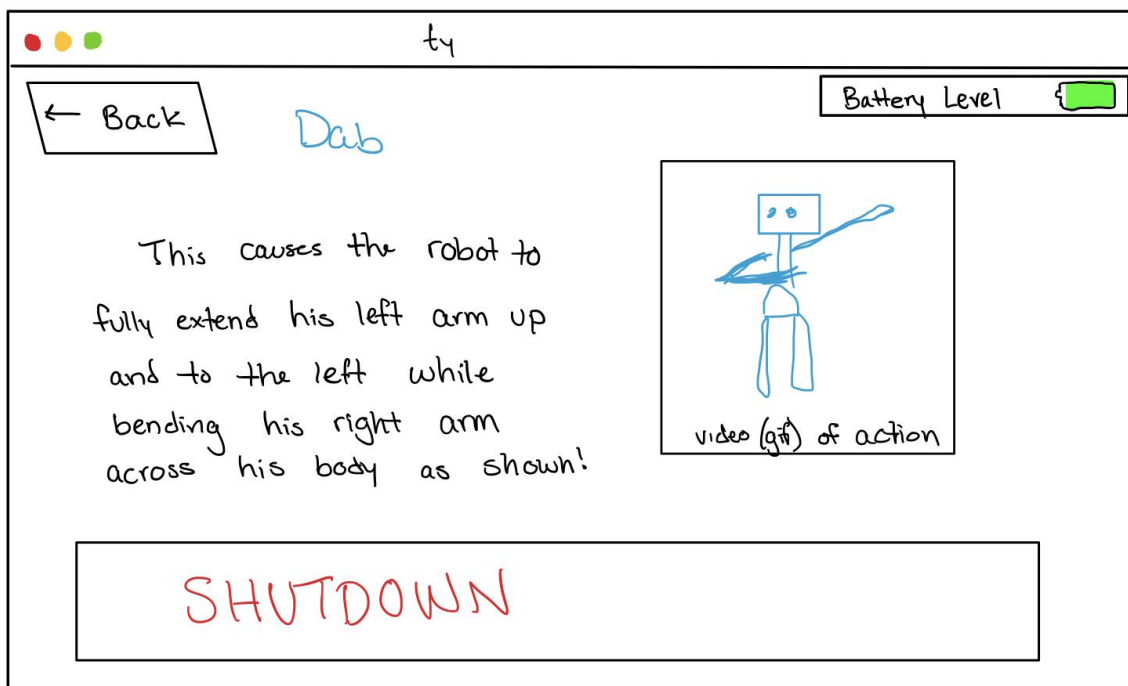


Figure 13.3: Second Iteration UI Prototype of Described Movements

After these edits were made, it was time for the team to design a mockup using HTML and CSS. This proved difficult, as some of our ideas were challenging to implement. For example, we had hoped the user would be able to swipe through different movement options, much like users can swipe through pictures on a mobile phone. However, the way to get this executed in HTML and CSS was too complex for a first iteration because it required a lot of stylistic code, so the team just decided to go with basic buttons. The first software iteration of the home page can be seen in Figure 13.4 below. This home page allowed users to get to the pre-recorded movements page or the record new movement page.

Koalby UI.

The robot has been initialized. Would you like to select prerecorded movements or record a new one?

Pre Recorded Movements

Record New Movement

?

SHUTDOWN

Toggle Sensor Data

Sensor Data

Battery Level is:

IMU is reading:

Figure 13.4: First Software UI Iteration

13.5 Final UI Design

Figures 13.5-13.7 show the final UI design. On each page, there was a navigation bar at the bottom, allowing users full control of where they want to go. Clicking the navigation bar brought users to the corresponding page. There were also various help buttons on every page in case a user gets lost. Additionally, back buttons were implemented throughout the application

allowing users to handle errors by themselves. Lastly, a red shutdown button is on the bottom of every page in case of emergencies.

The home page, Figure 13.5, shows the final version of the screen users see after initializing Koalby. There was a help button, describing the functionalities of each button on this page. There was a begin walking button, telling the robot to begin taking steps. If clicked, the UI displayed a message showing that the robot had begun walking, and the button changed to say stop walking. Clicking this told the robot to stop walking. Furthermore, the homepage featured an execute movement button, and a record new movement button bringing users to that specific page respectively. A show sensor data button is also there allowing users to view the IMU data and the battery level of the real robot.

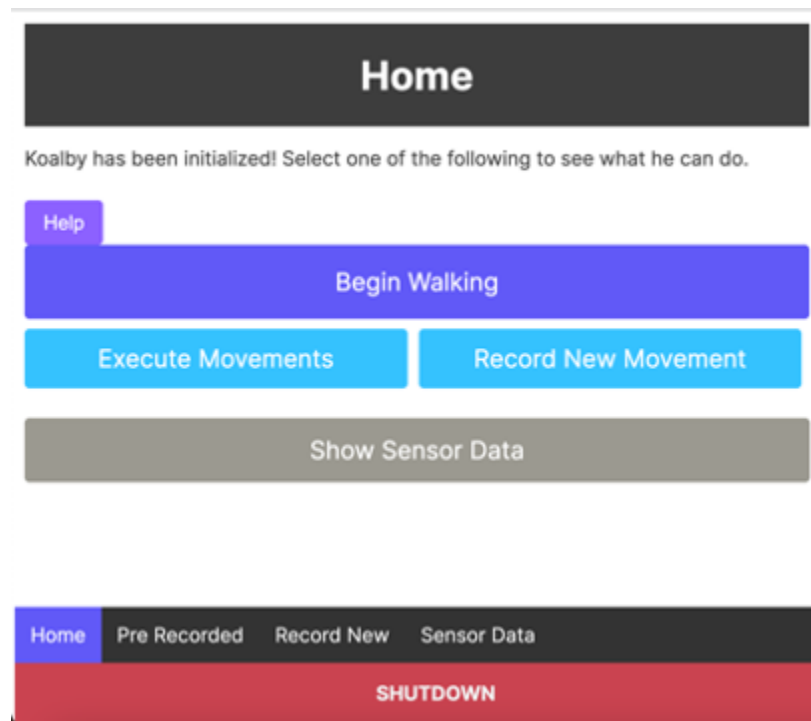


Figure 13.5: Final UI Homepage Design

The execute movements page, Figure 13.6, allowed the user to execute a list of pre-recorded movements. It displays a list of the executable movements. When clicked, they appear in the queue preview in order of selection. When the user was satisfied with their selections, they clicked run and these movements were executed in the same order. This page also features the back, shutdown, and sensor data buttons as mentioned.

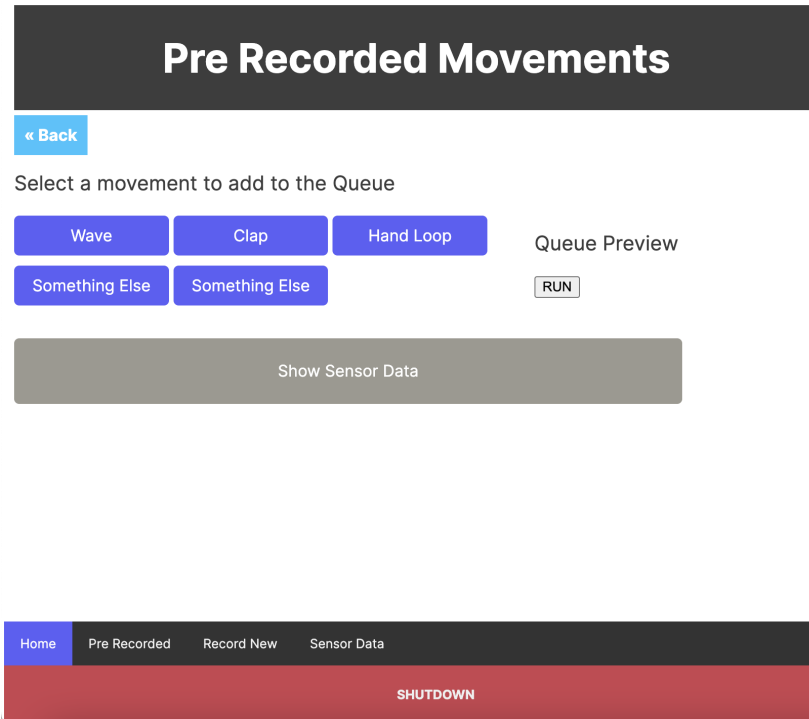


Figure 13.6: Pre Recorded Movements Page

The record new movements page, Figure 13.7, allowed the user to record their own movement. As previously mentioned, it contains back, help, and shutdown buttons for error prevention and handling. It then asked the user how many positions they would like to record and what to call this new movement. Then after clicking submit, a message was displayed saying the new file name and the amount of positions to record. Every time a press to record poses was clicked, the number decrements and the motor positions were recorded. When it reached 0, the button was disabled and a finished recording message was displayed. This new position could then be executed from the pre-recorded page.

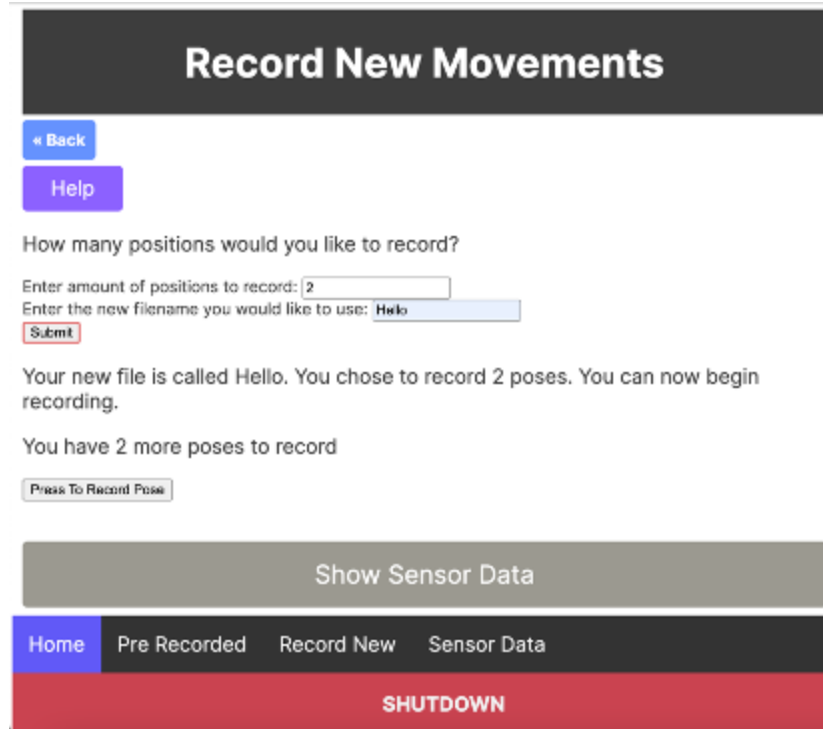


Figure 13.7: Final UI Record New Movement Design

Lastly, when the UI is being run on the real-world robot, the button to initialize the simulated robot is hidden on the welcome page, Figure 13.5. Errors were also prevented from the initialization, as the application prevented the user from moving forward if they were not connected to the robot. The color scheme was chosen with the robot's colors in mind, and was consistent throughout the application. The font was also the same for all text throughout the application.

14. Discussion

This chapter reviews the original goals of this project and how we addressed each one. Additionally, our live demonstration at WPI's Undergraduate Research Projects Showcase was reviewed, highlighting the key achievements and insights learned.

14.1 Goals

The main goals this team focused on achieving this year were to:

1. Assess and repair the physical robot
2. Calibrate and integrate motors and sensors
3. Recreate the robot and its movements in simulation
4. Develop code for stabilization and movement
5. Develop a user-friendly interface where even novice users could interact with the robot while seeing the sensor data in real time

14.2 Assess and Repair the Physical Robot

This project successfully assessed and then repaired Koalby. Based on our initial assessment, it was determined that several hardware components needed to be improved such as repairing broken parts, improving wire management, and redesigning the electrical circuits for new components (see Chapter 6.1). We worked with the 2023 3D Printed Humanoid Robot Design MQP Team to replace broken components, and we conducted major updates to the electrical components (see Chapter 9.4). These electrical updates included replacing the original solid core wires with higher quality threaded wires and resoldering weak connections. A new electrical diagram was constructed to accommodate new motors and sensors (see Appendix E). These changes included swapping out the two 7.4V batteries for stronger 11.1V batteries to power all of the additional components, and a series of adjustable voltage regulators were implemented to accommodate for the different voltages across sensors and motors.

Additionally, a software assessment was completed (see Chapter 6.2) which determined code needed to be refactored not only for useability and readability reasons, but also so we could implement more features such as sensors and a new UI. We worked on useability and readability

first, keeping a consistent naming convention and removing unnecessary threads. Then as we implemented new features, we updated the existing code to integrate the new features (see Chapter 12). Our next steps for the software would be to do more refactoring to consolidate files, and remove duplicated code. We also could make it easier to implement new features and sensors by abstracting the sensors and movement features.

14.3 Calibrate and Integrate Motors and Sensors

Overall, the sensor integration was mostly successful. The team was able to choose various sensors to aid in stabilizing Koalby. Code was sourced and developed, which was then tested. One of the improvements added was implementing a filter to the IMU raw data in order to mitigate sensor noise (see Chapter 11.3). We were also able to run various tests to familiarize ourselves with each sensor and know its limitations (see Chapter 9.3).

However, the sensor integration was not fully successful. While all of the foundations for each sensor were in place, they were not able to be incorporated into the robot's functionality. There was code for PI control, but it was not used in getting the robot standing stable. It was also difficult to integrate the TF Luna code into the codebase. It worked fine on its own separate project, but then when integrated into the main Koalby code, it read values that were nonsense. Furthermore, little testing was done to incorporate the Huskylens code into the main code base. The next steps in this aspect of the project would be to further integrate and test the sensors into Koalby's functionality.

Electronics were successful to a degree as the team was able to standardize the brand of motors and keep the cost low, by replacing the Dynamixel motor with HerkuleX motors (see Chapter 9.2). New battery circuits were developed which incorporated the new sensors and motors along with the necessary voltage required to run all the components (see Chapter 9.4). The wiring proved to be a big hurdle for the team due to old, solid core, wire that kept breaking or falling out of pins, this issue was addressed throughout the term and the worn components were replaced with higher quality, threaded wires, to ensure consistency. This proved to be successful as Koalby was able to be turned on and off repeatedly for over five hours WPI's Undergraduate Research Projects Showcase with no wire issues.

14.4 Recreate the Robot and its Movements in Simulation

This team successfully created the robot in simulation (see Chapter 10) and developed standing and walking in simulation (see Chapter 11). In CoppeliaSim, we successfully developed a stable standing position with and without PI control. Simulation stability trials with integrated sensors determined that Koalby could stand for an average of 15 seconds longer with the PI control loop enabled. After the standing positions were developed in simulation, they were applied to the real world. This project was able to have Koalby stand for upward of 20 minutes with his hands attached to the cart. This was done with the same motor positions developed in simulation to balance. With this balanced position, the team was able to run the primitive actions of waving and shaking hands while Koalby stood with one hand attached to the cart. These motions were successfully demonstrated at the WPI Undergraduate Research Projects Showcase (see Chapter 14.7). Due to time constraints, limited testing of unassisted standing was completed and the PI control was not integrated into the actual robot. The most successful unassisted standing test resulted in the robot standing in the same simulated balanced position while being held from the pelvis by one finger. The next steps would be to integrate the PI control into the actual robot and continue to test unassisted standing.

Additionally, assisted walking was developed in simulation. This assisted walking was done with Koalby's hands attached to a cart which he pushed (see Chapter 11.4). This project successfully developed the beginning 3-5 small steps of assisted walking with the cart in simulation. Also due to time constraints, limited testing of the trajectory planning was done on the actual robot. The walking motion was successfully tested on the robot while it was attached to the wooden support frame. The next steps would be developing the walking motion untethered from the support frame. This would require integrating the balanced standing position and PI control with the trajectory planning to develop walking in the real world.

14.5 Develop Code for Stabilization and Movement

This project successfully developed kinematics and trajectory planning for assisted walking in humanoid robotics (Chapters 7 and 8). The team was able to develop inverse and forward kinematics and trajectory planning to generate movement in the leg and arm. The leg movement was tested in simulation and on the actual robot. The arm code was developed and

tested to ensure the final position matched the initial position. These functionalities were achieved in order to add more robust and controlled motions such as walking and grasping objects.

For future work, the timing in the trajectory planning will need to be optimized to create a more natural movement. The arm code is currently in MATLAB and future teams should implement it into the overall Python code. Due to timing constraints and missing components, future teams can run tests using the arm code base.

14.6 Develop a User-Friendly Interface

The team was successfully able to create a user friendly interface with capabilities for full control of the robot (see Chapter 13). All buttons are functional (except for disco lights), and were tested in the simulation and in real life. Additionally, other buttons are in place, such as those for reaching objects, so that once more functionalities are added to the robot, these buttons can be enabled and functional.

For future work, the UI could be improved upon in various ways. More user studies and tests can be conducted in order to make it better. It could also be improved for mobile use, as some of the stylistic designs are not currently consistent.

14.7 Demonstration at Undergraduate Research Projects Showcase

This project was demonstrated at WPI's Undergraduate Research Projects Showcase. This event allowed the team to successfully demonstrate and present the project at a higher level, speaking to WPI faculty and engineering professionals. Koalby successfully demonstrated assisted walking with the cart and performed primitives such as waving and shaking hands. This team presented in the Mechanical and Materials Engineering Department and Robotics Engineering Department for a total of about five hours, and Koalby had no mechanical, software, or electrical failures the entire day, except for the batteries dying during the fourth hour of demonstrations. Figure 14.1 shows this team's poster presentation at WPI's Undergraduate Research Projects Showcase.



Figure 14.1: Mechanical and Materials Engineering Department Presentation

15. Conclusions

Overall, this team successfully developed Koalby towards the application as lab assistant with our three main goals. First, this team successfully simulated standing and assisted walking in Coppeliasim, and then applied this to the real-world with assisted standing while performing primitives. This accomplishment required the development of codes for stabilization and walking including a PI controller, forward and inverse kinematics, and trajectory planning. These codes were developed individually and then integrated together into the Koalby interface.

Beyond the initial objective to develop standing and assisted walking in a humanoid robot, a user-friendly interface was created to allow for ease in controlling the robot's motion and displaying sensor data. The UI is capable of playing pre-recorded movements, recording new movements, starting and stopping walking, and displaying sensor data. Additionally, the team investigated and implemented different sensors and electronics into the system. Three new sensors, IMU, LiDAR TF Luna, and AI Huskylens camera, were independently tested and integrated in simulation.

15.1 Broader Impacts

15.1.1 Engineering Ethics

This project followed the Mechanical Engineering Code of Ethics. First, we used our knowledge and skill for the enhancement of human welfare. By developing standing and assisted walking capabilities in a humanoid robot, we worked towards the application of a humanoid robot lab assistant which would be used to assist people in a lab setting. This assistance contributes directly to improving human welfare by helping people. Second, we were honest and impartial throughout the processes of this project. Throughout this project, we took care in accurately documenting our progress, including our achievements and limitations. Third, we strived to increase the competence and prestige of the engineering profession. Bipedal locomotion in humanoid robots is a current engineering challenge where solutions are still being developed. This project contributed to understanding the code and sensors required for walking and successfully applying it in simulation.

15.1.2 Societal Impact

This project and its achievements contribute to larger societal impacts in the fields of service. By developing standing and walking in a humanoid robot, we worked towards the larger application of creating a lab assistant. As a lab assistant, Koalby can be utilized to aid people in various lab activities such as pushing a cart of supplies and picking up tools. This can improve the work load for lab staff by eliminating menial tasks required by them, allowing for more specialized focus. Furthermore, the functionalities of standing and walking in humanoid robots can be expanded to more applications such as medicine, industry, and academia (as discussed in Chapter 4.1). Additionally, this project created a user-friendly interface which made the robot's controls significantly more accessible (see Chapter 13). This UI allows non-experienced users to control the robot which expands the domain of possible users. Koalby is no longer limited to highly specialized, experienced users to operate.

15.2 Recommendations for Future Work

15.2.1 Autonomy

With all of the sensor functionality in place, we hope that future teams are able to integrate all of this functionality into the robot's main codebase, allowing Koalby to make decisions to pick objects up or walk/stop depending on feedback from its different sensors. For example, when the TF Luna and Huskylens are integrated, along with inverse kinematics and the gripper, Koalby can use their feedback to pick up a specified object and will be able to move his arm and hand to the location of the object, and pick it up, with one button press from the user.

15.2.2 Optimization

On the software side, optimization can be done to reduce processing time. As mentioned in Chapter 12.2, it takes about 6.66 seconds from when the user clicks run to run "RunToInitializeKoalby" to when the real robot actually initializes. This has to do with the slow serial communication between the processor running the python code and the arduino. Additionally, in a lot of places, the code is forced to wait due to the communication delay. There are ways to speed up this process. The first is sending a smaller amount of data. In order to do

this, a future team could send data as binary instead of sending it as an ASCII value. In general, binary files occupy less space on a disk than their ASCII equivalents. While this would speed the actual transfer of data, the python would also then have to decipher the binary when it is received. As this has not been tested yet, new tests would have to be conducted to see if this is faster than it already is. Currently, ASCII values are being passed to and from the Python. Additionally, future teams could use a higher baud rate as that would theoretically speed up the rate that information is transferred.

Furthermore, the general processing time of the code can be improved. Loops are widely used throughout the code base, and when processing large amounts of data, can be slow. These loops could be refactored and made into something such as a stream to speed up the process. Additionally, the code could be refactored to use various design patterns. These design patterns can be used for scalability and accessibility for future teams.

15.2.3 Increasing Capabilities

This robot still requires further testing and integration of components to increase its capabilities. Related next steps include integrating the sensor codes and hardware into Koalby's interface, so that capabilities of Koalby in simulation can be fully transferred to the real world. For the robot to be used as a lab assistant, the various sensors need to be able to work together to stabilize Koalby and provide information while walking and picking up objects. Once this integration occurs and Koalby is capable of walking in the real-world, more movements can be developed specific to picking up objects and working as a lab assistant.

References

- [1] A. Gasparetto and L. Scalera, "A brief history of industrial robotics in the 20th Century," *Advances in Historical Studies*, 31-Jan-2019. [Online]. Available: <https://www.scirp.org/journal/paperinformation.aspx?paperid=90517>. [Accessed: 27-Apr-2023].
- [2] N. G. Hockstein, C. G. Gourin, R. A. Faust, and D. J. Terris, "A history of robots: From science fiction to Surgical Robotics - Journal of Robotic surgery," *SpringerLink*, 17-Mar-2007. [Online]. Available: <https://link.springer.com/article/10.1007/s11701-007-0021-2>. [Accessed: 28-Apr-2023].
- [3] T. Fukuda, P. Dario, and G.-Z. Yang, "Humanoid robotics—history, current state of the art, and challenges," *Science Robotics*, 20-Dec-2017. [Online]. Available: <https://www.science.org/doi/10.1126/scirobotics.aar4043>. [Accessed: 28-Apr-2023].
- [4] S. Behnke, "Humanoid Robots - From Fiction to Reality?," *ResearchGate*, Jan-2008. [Online]. Available: https://www.researchgate.net/publication/220634191_Humanoid_Robots_-_From_Fiction_to_Reality. [Accessed: 28-Apr-2023].
- [5] M. M. E. van Pinxteren, R. W. H. Wetzels, J. Rüger, M. Pluymaekers, and M. Wetzels, "Trust in humanoid robots: Implications for services marketing," *Journal of Services Marketing*, 28-Jan-2019. [Online]. Available: <https://www.emerald.com/insight/content/doi/10.1108/JSM-01-2018-0045/full/html>. [Accessed: 28-Apr-2023].
- [6] "The story behind the Poppy Project," *Poppy Project - About*. [Online]. Available: <https://www.poppy-project.org/en/about/>. [Accessed: 28-Apr-2023].
- [7] A. Galgano, D. Fournet, A. Lehman, W. Engdahl, and R. Beazley, "3D printed humanoid Robot Project," *Digital WPI*, 28-Apr-2022. [Online]. Available: https://digital.wpi.edu/concern/student_works/dr26z168n?locale=en. [Accessed: 28-Apr-2023].

- [8] “Visualize the Robot in Simulation,” *Visualize · Documentation of the Poppy Platform*. [Online]. Available: <https://docs.poppy-project.org/en/getting-started/visualize.html>. [Accessed: 27-Apr-2023].
- [9] “Open Source Technologies for Building Amazing Robots,” *Poppy Project - Technologies*. [Online]. Available: <https://www.poppy-project.org/en/technologies/>. [Accessed: 27-Apr-2023].
- [10] R. Beazley, W. Engdahl, D. Fournet, A. Galgano, and A. Lehman, “3D Printed Humanoid Robot.” Worcester Polytechnic Institute, May-2022.
- [11] D. Merkusheva, “Top 10 Examples of Humanoid Robots,” *ASME*, 25-Mar-2020. [Online]. Available: <https://www.asme.org/topics-resources/content/10-humanoid-robots-of-2020#:~:text=Humanoid%20robots%20are%20used%20for,%2C%20public%20relations%2C%20and%20healthcare.> [Accessed: 22-Nov-2022].
- [12] Agility Robotics “Robots,” *Agility Robotics*. 2022. [Online]. Available: <https://agilityrobotics.com/robots>. [Accessed: 22-Nov-2022].
- [13] Kawada Robotics, “Concept,” *NEXTAGE*, 2022. [Online]. Available: <http://nextage.kawada.jp/en/concept/>. [Accessed: 22-Nov-2022].
- [14] Macco Robotics, “Kime: Macco: Robotics Technology for hospitality: Spain,” *Maccorobotfoodtech*, 2022. [Online]. Available: <https://www.maccorobotics.com/robot-camarero-kime?lang=en>. [Accessed: 22-Nov-2022].
- [15] Hanson Robotics, “Sophia,” *Hanson Robotics*, 01-Sep-2020. [Online]. Available: <https://www.hansonrobotics.com/sophia/>. [Accessed: 22-Nov-2022].
- [16] E. Guizzo, “Iran unveils its most advanced humanoid robot yet,” *IEEE Spectrum*, 18-Aug-2022. [Online]. Available: <https://spectrum.ieee.org/iran-surena-iv-humanoid-robot>. [Accessed: 22-Nov-2022].
- [17] “What are D-H parameters?,” *Marginally Clever Robots*, 05-Jan-2022. [Online]. Available: <https://www.marginallyclever.com/2020/04/what-are-d-h-parameters/>. [Accessed: 04-Apr-2023].

- [18] M. Agheli (2021). RBE 3001 Lectures 6-10 on Forward and Inverse Kinematics [PowerPoint slides]. [Accessed: 04-Apr-2023].
- [19] A. Rosendo (2023). RBE 4815 Lectures 7-8 on PoE [PowerPoint slides]. [Accessed: 06-Apr-2023].
- [20] M. Agheli (2021). RBE 3001 Lectures 11-12 on Trajectory Planning [PowerPoint slides]. [Accessed: 04-Apr-2023].
- [21] V. Mazzari, "IMU and robotics: All you need to know," *Génération Robots - Blog*, 29-Mar-2023. [Online]. Available: <https://www.generationrobots.com/blog/en/imu-and-robotics-all-you-need-to-know-2/>. [Accessed: 02-May-2023].
- [22] G. Lewin (2021). RBE 3001 Lectures 04/16 Intro to the IMU [PowerPoint slides]. [Accessed: 04-Apr-2023].
- [23] M. Looney, "Anticipating and managing critical noise sources in MEMS gyroscopes," *Anticipating and Managing Critical Noise Sources In MEMS Gyroscopes | Analog Devices*. [Online]. Available: [https://www.analog.com/en/technical-articles/critical-noise-sources-mems-gyroscopes.html#:~:text=The%20inherent%20sensor%20noise%20represents,nois%C%20with%20respect%20to%20frequency](https://www.analog.com/en/technical-articles/critical-noise-sources-mems-gyroscopes.html#:~:text=The%20inherent%20sensor%20noise%20represents,nois%C%20with%20respect%20to%20frequency.). [Accessed: 27-Apr-2023].
- [24] "Understanding Sensor Bias (offset)," *Xsens knowledge base*, 21-Feb-2022. [Online]. Available: https://base.xsens.com/s/article/Understanding-Sensor-Bias-offset?language=en_US. [Accessed: 27-Apr-2023].
- [25] P. Secor, "Why robots need to see - RGO robotics blog," *RGo Robotics*, 21-Apr-2022. [Online]. Available: [https://www.rgorobotics.ai/post/why-robots-need-to-see#:~:text=For%20example%2C%20because%20they%20can,the%20way%20a%20human%20would](https://www.rgorobotics.ai/post/why-robots-need-to-see#:~:text=For%20example%2C%20because%20they%20can,the%20way%20a%20human%20would.). [Accessed: 27-Apr-2023].

- [26] “3 ways robots see the world,” *Boston Dynamics*, 07-Apr-2022. [Online]. Available: <https://www.bostondynamics.com/resources/blog/3-ways-robots-see-world>. [Accessed: 02-May-2023].
- [27] K. Gremillion, “Choose the right sensors for Autonomous Vehicles,” *Semiconductor Engineering*, 09-Dec-2021. [Online]. Available: <https://semiengineering.com/choose-the-right-sensors-for-autonomous-vehicles/#:~:text=4.,Ultrasonic,reflected%20back%20to%20the%20sensor>. [Accessed: 27-Apr-2023].
- [28] L. Wasser, “The Basics of Lidar - Light Detection and Ranging - Remote Sensing,” *Open Data to Understand our Ecosystems*. [Online]. Available: <https://www.neonscience.org/resources/learning-hub/tutorials/lidar-basics>. [Accessed: 27-Apr-2023].
- [29] H. Webmaster, “Advantages of 3D vs. 2D lidar in AGV Applications,” *HESAI*, 20-Apr-2023. [Online]. Available: <https://www.hesaitech.com/advantages-of-3d-vs-2d-lidar-in-agv-applications/#:~:text=The%20key%20difference%20between%202D,dimensional%20view%20of%20the%20area>. [Accessed: 02-May-2023].
- [30] “What is Lidar and why lidar,” *LeddarTech*, 17-Feb-2022. [Online]. Available: <https://leddarsensor.com/why-lidar/>. [Accessed: 27-Apr-2023].
- [31] Admin, “Lidar and Radar Information,” – *LiDAR and RADAR Information*. [Online]. Available: <https://lidarradar.com/info/advantages-and-disadvantages-of-lidar>. [Accessed: 27-Apr-2023].
- [32] D. Jost, “What is an ultrasonic sensor?,” *Fierce Electronics*, 07-Oct-2019. [Online]. Available: <https://www.fierceelectronics.com/sensors/what-ultrasonic-sensor>. [Accessed: 27-Apr-2023].
- [33] L. Reese, “The working principle, applications and limitations of ultrasonic sensors,” *Microcontroller Tips*, 06-Aug-2019. [Online]. Available:

<https://www.microcontrollertips.com/principle-applications-limitations-ultrasonic-sensors-faq/>.
[Accessed: 27-Apr-2023].

[34] “15 applications for ultrasonic sensors: Migatron Corp.,” *Ultrasonic Sensors | Migatron Corp.*, 15-Mar-2018. [Online]. Available:

<https://www.migatron.com/ultrasonic-detections-and-control-applications/#:~:text=There%20are%20two%20types%20of%20ultrasonic%20sensors&text=The%20detect%20point%20is%20independent,reflected%20bursts%20of%20ultrasonic%20sound>. [Accessed: 02-May-2023].

[35] “Ultrasonic sensor advantages,” *Ixthus*, 2022. [Online]. Available:

<https://www.ixthus.co.uk/news-media/blog-archive/ultrasonic-sensor-advantages>. [Accessed: 27-Apr-2023].

[36] T. Islam, M. S. Islam, M. Shajid-Ul-Mahmud, and M. Hossam-E-Haider, “Comparison of complementary and Kalman filter based data fusion for attitude heading reference system,” *AIP Conference Proceedings*, 2017.

[37] H. Liu, C. Luo, and L. Zhang, “Target recognition and Heavy Load Operation Posture Control of humanoid robot for trolley operation,” *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, 2018.

[38] D. Bazylev and A. Pyrkin, "Stabilization of biped robot standing on nonstationary plane," 2013 18th International Conference on Methods & Models in Automation & Robotics (MMAR), Miedzyzdroje, Poland, 2013, pp. 459-463, doi: 10.1109/MMAR.2013.6669952.

[39] G. H. Liu, M. Z. Chen, and Y. Chen, “When joggers meet robots: The past, present, and future of research on Humanoid Robots,” *Bio-Design and Manufacturing*, vol. 2, no. 2, pp. 108–118, 2019.

[40] K. Yin, Y. Xue, Y. Yu, and S. Xie, “Variable impedance control for bipedal robot standing balance based on artificial muscle activation model,” *Journal of Robotics*, vol. 2021, pp. 1–9, 2021.

[41] S. Zhuge, “PID Control Theory,” *Sentek Dynamics*, 05-Feb-2021. [Online]. Available:

<https://www.sentekdynamics.com/sentek-dynamics-news/2020/8/24/pid-control-theory?gclid=Cj>

0KCQiAvqGcBhCJARIsAFQ5ke6NK7eFAd2bBkIAwbnJEUXteeefP4YT0tzYErXG3TyZihNtT
BsPD0IaApBIEALw_wcB. [Accessed: 27-Apr-2023].

[42] J. C. Vaz and P. Oh, “Material Handling by Humanoid Robot While pushing Carts Using a Walking Pattern Based on Capture Point,” 2020 IEEE International Conference on Robotics and Automation, Aug-2020. [Online]. Available: https://drive.google.com/file/d/1u6poTADYp34d-K7pRSK_tYL4Oo1kvy8B/view. [Accessed: 09-Oct-2022].

[43] “What is heuristic evaluation?,” The Interaction Design Foundation, 15-Jun-2021. [Online]. Available: <https://www.interaction-design.org/literature/topics/heuristic-evaluation>. [Accessed: 26-Apr-2023].

[44] J. Nielsen and R. Molich, “Heuristic Evaluation of User Interfaces.” Association for Computing Machinery, New York, NY, 1990.

[45] E. Solovey, “HEURISTIC EVALUATION,” 2022.

[46] A. Lodhi, “2010 2nd International Conference on Software Technology and Engineering,” in *Usability Heuristics as an assessment parameter: For performing Usability Testing*, vol. 2, pp. 256–259.

[47] W.-siong Tan, D. Liu, and R. Bishu, “Web Evaluation: Heuristic Evaluation vs. User Testing,” *International Journal of Industrial Ergonomics*, vol. 39, no. 4, pp. 621–627, Jul. 2009.

[48] Leon Žlajpah, Abstract Simulation has Been Recognized as an Important Research Tool Since the Beginning of the 20th Century. A. Cavalcanti, P. I. Corke, S. Dubowsky, C.-M. Éve, J. Go, H. Hirukawa, O. Khatib, A. T. Miller, and A. Miller, “Simulation in robotics,” *Mathematics and Computers in Simulation*, 16-Feb-2008. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0378475408001183>. [Accessed: 26-Apr-2023].

[49] “UUV Simulator: A Gazebo-based Package for Underwater Intervention and Multi-Robot Simulation.” [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/7761080/>. [Accessed: 27-Apr-2023].



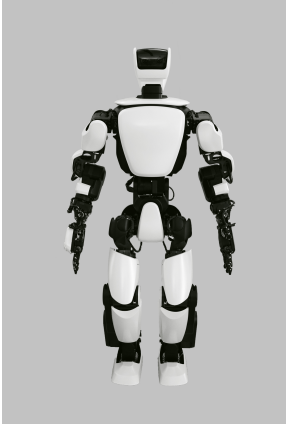

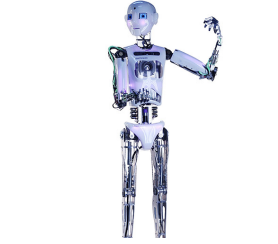
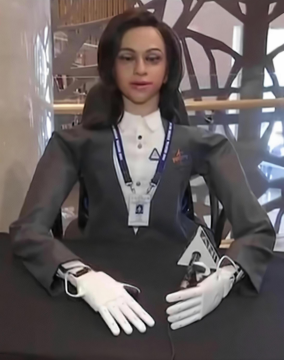
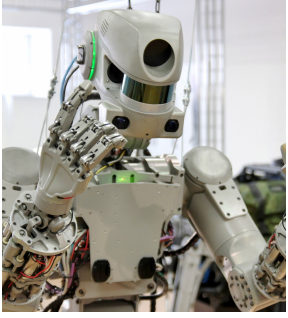
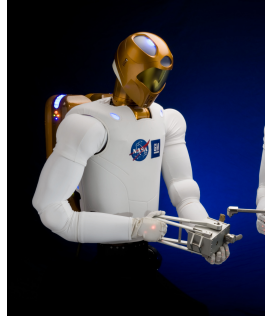

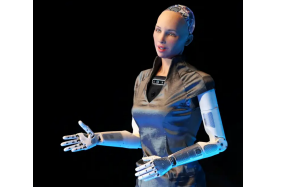

[50] “The Trifecta: Great Concept, Lousy Statistic.” [Online]. Available: <https://bjui-journals.onlinelibrary.wiley.com/doi/full/10.1111/j.1464-410X.2012.11327.x?cookieSet=1>. [Accessed: 27-Apr-2023].

[51] “V-REP: A Versatile and Scalable Robot Simulation Framework.” [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6696520/>. [Accessed: 27-Apr-2023].

[52] A. Industries, “Adafruit 9-DOF absolute orientation IMU Fusion Breakout - BNO055,” *adafruit industries blog RSS*. [Online]. Available: <https://www.adafruit.com/product/2472>. [Accessed: 02-May-2023].

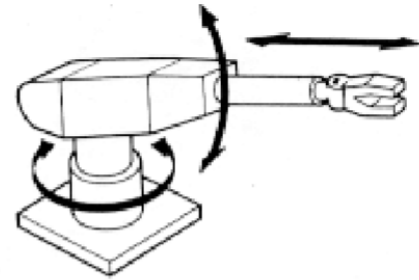
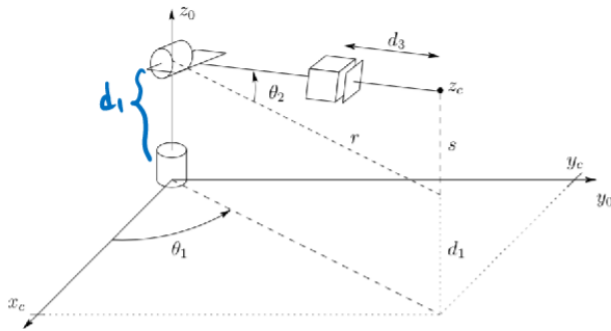
Appendices

Appendix A: Humanoid Applications

Digit (Ford Agility Robotics)	Nextage (Kawada Robotics)	T-HR3 (Toyota)	Kime (Macco Robotics)
			
Robotthespian	Vyammitra	Fedar	Robonaut 2 (NASA)
			
Valkyrie (NASA)	Sophia (Hanson Robotics)	Surena Robot (Iranian U)	
			

Appendix B: Kinematics and Trajectory Planning

Kinematics Calculations for Polar/Spherical Manipulators



Geometric Approach

$$r * \cos(\theta_1) = x_c \quad \rightarrow$$

$$\cos(\theta_1) = (x_c)/(r) = D \quad \rightarrow$$

$$\sin(\theta_1) = \pm \sqrt{(1 - D^2)}$$

$$\theta_1 = \text{atan2}(\pm \sqrt{(1 - D^2)}, D) \quad (1)$$

$$r^2 + s^2 = d_3^2 \quad \rightarrow$$

$$d_3 = \pm \sqrt{(r^2 + s^2)} \quad (2)$$

$$d_3 * \cos(\theta_2) = r \quad \rightarrow$$

$$\cos(\theta_2) = (r)/(d_3) = E \quad \rightarrow$$

$$\sin(\theta_2) = \pm \sqrt{1 - E^2}$$

$$\theta_2 = \text{atan2}(\pm \sqrt{(1 - E^2)}, E) \quad (3)$$

Algebraic Approach

$$1) x_c = d_3 * \cos(\theta_2) * \cos(\theta_1) \rightarrow \text{square: } x_c^2 = d_3^2 * \cos^2(\theta_2) * \cos^2(\theta_1)$$

$$2) y_c = d_3 * \cos(\theta_2) * \sin(\theta_1) \rightarrow \text{square: } y_c^2 = d_3^2 * \cos^2(\theta_2) * \sin^2(\theta_1)$$

$$3) z_c = d_1 + d_3 * \sin(\theta_2) \rightarrow \text{square: } (z_c - d_1)^2 = d_3^2 * \sin^2(\theta_2)$$

$$\text{sum: } G = x_c^2 + y_c^2 + (z_c - d_1)^2 =$$

$$d_3^2 * \cos^2(\theta_2) * \cos^2(\theta_1) + d_3^2 * \cos^2(\theta_2) * \sin^2(\theta_1) + d_3^2 * \sin^2(\theta_2) =$$

$$d_3^2 * \cos^2(\theta_2) + d_3^2 * \sin^2(\theta_2)$$

$$G = d_3^2 \rightarrow d_3 = \pm \sqrt{G} \quad (1)$$

$$z_c - d_1 = d_3 * \sin(\theta_2) \rightarrow \sin(\theta_2) = (z_c - d_1)/d_3 = C \rightarrow \cos(\theta_2) = \pm \sqrt{1 - C^2}$$

$$\theta_2 = \text{atan2}(C, \pm \sqrt{1 - C^2}) \quad (2)$$

$$x_c = d_3 * \cos(\theta_2) * \cos(\theta_1) \rightarrow \cos(\theta_1) = (x_c)/(d_3 * \cos(\theta_2)) = F \rightarrow \sin(\theta_1) = \pm \sqrt{1 - F^2}$$

$$\theta_1 = \text{atan2}(\pm \sqrt{1 - F^2}, F) \quad (3)$$

Appendix C: Troubleshooting Document

 Troubleshooting

[Troubleshooting Document](#)

Appendix D: GitHub and Simulation

Github:

<https://github.com/orgs/KoalbyMQP22-23/repositories>

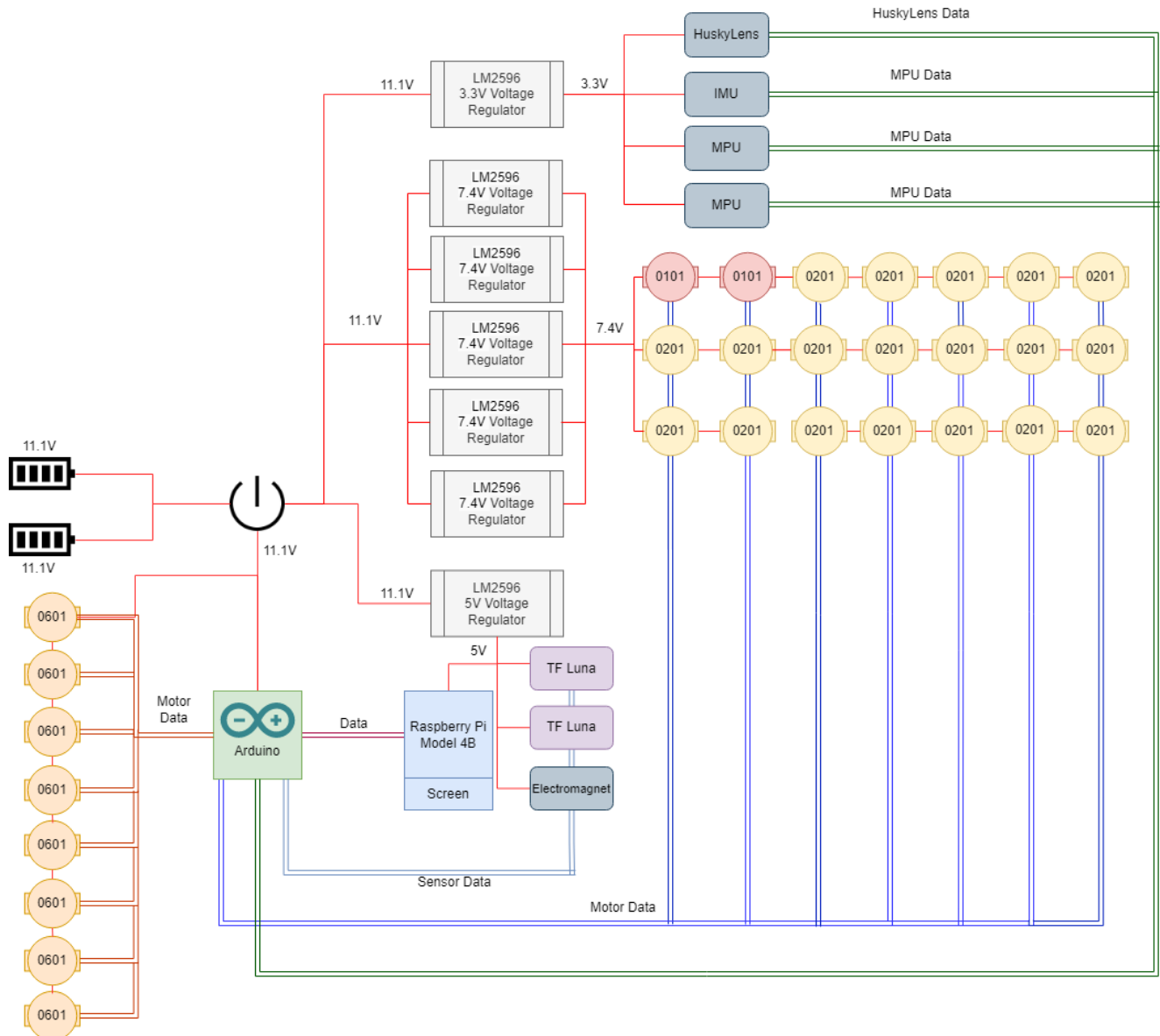
The github contains several repositories, each of which is documented internally:

- Flask Project
 - Rewritten Python code to control the robot, interfaces with firmware
 - Contains everything from simulation to UI
- Arduino Code
 - Arduino firmware, receives commands from Python code
 - Testing programs, used at various points to verify features of the motors
 - Setup programs, assign motor ID's and read motor positions to set limits
 - Sensor programs - base programs to read sensor data
- RaspberryPi
 - Fork of Flask-Project without simulation files to be run on the Pi
- Koalby Humanoid
 - Fork from last year's team - aborted when flask application was created
- MATLAB Code

To set up Koalby's software, go to the GitHub link above and click on the green "Code" button that appears on the page. After, click on the repository you want; "flask-project" is the Python code, "raspberrypi" is the code for the Raspberry Pi, and "Arduino-Code" is the arduino code component. The repository "koalby-humanoid" is an as-of-now outdated version of the Python code, so use "flask project" instead for that. You can then hit the option to download the .ZIP file, extract it, and open the source folder in your code environment. You can also hit the option to clone the repository, copy the link and/or open it directly using GitHub desktop. Once the code is open on your computer, each of the main files are titled "run this to..." and each controls a different testing component that can be run. The other supplementary files in the repository should be self-explanatory with their titling.

To set up the simulation environment, in the “flask-project” repository there is a folder in “backend” → “KoalbyHumanoid” that contains three “.ttx” files. These are scenes that you can open within CoppeliaSim by clicking “File” then “Open Scene.” The “Koalby Skeleton Sim bent arms cart.ttx” file is the primary one to test Koalby’s walk cycle and the “Koalby Skeleton Sim T Pose.ttx” file is the primary one to test Koalby standing without assistance with PI control. There is also a “Koalby Skeleton Sim New Arms.ttx” file for testing the inverse kinematics of the arms.

Appendix E: Electrical Diagram



Reflections

This project required two valuable interpersonal skills from each team member to achieve all that we did throughout this year: communication and organization. This Major Qualifying Project required 15-17 hours of work per week from each member, so the ability to communicate effectively and timely between team members was crucial to the project's success. Strong organizational skills were required to delegate tasks and document our progress.

Additionally, various technical skills were brought to this project from individual team members. Our team members had a strong technical background various areas, ranging from kinematics, to full stack software applications, to mechanical design. Over the course of this project, our team learned from each other while also developing new skills related to electrical components and sensor integration.

Overall, this project successfully continued the 2022 3D Printed Humanoid MQP Team's work by developing standing and assisted walking in simulation that was then applied to the real world with the successful demonstrations of assisted standing and waving.