



3D Printed Humanoid Robot

IN PARTIAL FULFILLMENT OF A MAJOR QUALIFYING PROJECT
WORCESTER POLYTECHNIC INSTITUTE

Submitted by:

Raymond Beazley (RBE/ME)

William Engdahl (RBE/ME)

David Fournet (RBE)

Anthony Galgano (RBE/ECE)

Alexandria Lehman (ME)

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

Research on humanoid robots often involves prohibitively expensive hardware. Using modern manufacturing techniques and off-the-shelf components, a small-scale humanoid robot can be manufactured at a much lower price point while still demonstrating many of the same capabilities as larger robots. Such robots can be used in many different applications including rehabilitation, research, and teaching. This paper presents a 4 kg, 85 cm tall humanoid robot with 27 degrees of freedom based on the open-source Poppy Project. The robot is constructed from DLP/SLA printed components that are easily modified to support alternative motor choices. Resin 3D printing is used to minimize weight via pocketing and allows types of resin to be mixed in ratios based on whether a part needs stiffness or overall strength. Each joint is controlled by one of several types of smart actuators with integrated position control. The Poppy project exclusively uses 25 Dynamixel motors, which make up the bulk of the robot's cost. We replaced most of these motors with the less expensive HerkuleX DRS-0201 motors that have comparable form factor and power to the Dynamixel MX-28. Physical changes included adaptors for the HerkuleX motors to mount to the preexisting mounting patterns.

Pivoting away from the Poppy Project's exclusive use of the more expensive Dynamixel motors required the codebase to be written from scratch. An onboard Arduino communicates with the different actuators, while a Raspberry Pi performs higher level processing. The Arduino supports multiple actuators using different communication buses. It provides a layer of abstraction allowing the Raspberry Pi to control motors without accounting for the specific actuator or motion limits at each joint. The robot now uses batteries to allow for completely untethered operation. It is able to walk with human assistance but has the capability for future implementation of self-balancing for unassisted walking. Control is based on a primitive system that allows multiple actions to occur simultaneously. This allows either multiple actions using different parts of the body (walking and waving arm), or by averaging positions to combine actions utilizing the same part of the body (a normal walking gait and adjustments to terrain based on sensor feedback). We demonstrate the robot's functionality using a record/play system that allows a human operator to manually position the robot and then record those positions for later playback along with functions that highlight the capabilities of individual systems of the robot. The paper will describe the implementation, challenges faced, and future work to further added capabilities.

Acknowledgements

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

Dr. Pradeep Radhakrishnan

Dr. Kaveh Pahlavan

Barbara Fuhman

Peter Hefti

The original Poppy Project team at the Flowers laboratory at Inria Bordeaux Sud-Ouest

Worcester Polytechnic Institute

Our friends and families

Abstract	1
Acknowledgements	2
1 Introduction	6
1.1 Poppy	8
1.2 Poppy Redesigns	13
1.3 Objectives	15
1.4 Section Overview	15
2 Literature Review	16
2.1 Humanoid Gait	16
2.2 Actuated Hands	17
3 Project Plan and Methodology	20
3.1 Motor Selection	20
3.2 Redesigning for HerkuleX	21
3.3 Printing the Structure	22
3.3.1 FDM Printing	22
3.3.2 Resin Printing	23
3.4 Using the HerkuleX motors	24
3.5 Programming Koalby	24
3.6 Gantt Chart Details	25
3.7 Term Progress	27
3.7.1 A Term - Research and Purchasing	27
3.7.2.B Term - Initial Assembly	27
3.7.3 C Term - Code and Revisions	28
3.7.4 D Term - Finalizing Year's Progress	28
4 Design Changes	29
4.1 Design Modifications for HerkuleX	29
4.1.1 Motor Adapter	29
4.1.2 Motor Double Rotation	31
4.1.3 Arm	32
4.1.4 Torso	33
4.1.5 Lower Limbs	33
4.2 Mechanical Additions	35
4.2.1 Shin and Foot	35
4.2.2 Grasping Hands	38
4.3 Electronic Design Changes	40

4.3.1 Batteries	40
4.3.2 Control Boards	41
5 Manufacturing and Assembly	43
5.1 3D Printing	43
5.2 Laser Cutting	45
5.3 Assembly	47
6 Electronics	48
6.1 Electrical Power	48
6.2 Electronic Control	49
7 Coding	53
7.1 Arduino Firmware	53
7.2 Python Control Code	56
7.2.1 Pi/Arduino Communication	56
7.2.2 Robot Kinematics	58
7.2.3 Koalby Primitive System	59
7.2.4 Record / Replay Primitive	60
7.2.5 Multi-Threading and Test Files	61
7.2.6 User Interface	62
8 Testing	64
8.1 Arm Testing	64
8.2 Routine Testing	65
8.3 Battery Endurance Testing (TouchTomorrow)	67
9 Discussion	68
9.1 Goals	68
9.2 Mechanical Design and Construction of the Robot	68
9.3 Battery Power	69
9.4 Cost Reduction	70
9.5 Walking	71
9.6 Grasping	71
9.7 Demonstration at TouchTomorrow 2022	71
9.8 Demonstration at Project Presentation Day	73
10 Conclusions	74
10.1 Broader Impacts	74
10.1.1 Engineering Ethics	74
10.1.2 Societal Impact	75
10.2 Future work	75

10.2.1 Hardware	75
10.2.1.1 Design Simplification	75
10.2.1.2 Motor Selection	77
10.2.1.3 Grasping hands	79
10.2.2 Software	79
10.2.2.1 Sensing	79
10.2.2.2 Kinematics	79
10.2.2.3 Self-Balancing	80
10.3 Project Experience	80
11. References	84
12. Appendices	88
Appendix A: 3D printing (resin) Guide	88
Appendix B: Cost Breakdown Between Poppy and Koalby	89
Appendix C: Setup of Github/Grabcad/Google Drive	90
GrabCAD:	90
Github:	91
Appendix D: Assembly Instructions	91
Appendix E: Raspberry Pi Setup Instructions	92
Appendix F: Shin Torque Calculations and Initial Arm Testing	92
Appendix G: Cost and Time Breakdown of DLP Resin Printed Parts	93
Appendix H: Koalby Demonstration Videos	95
Appendix I: Team Contact Information	96

1 Introduction

A robot is a machine capable of carrying out a complex set of tasks, either with an external or internal controller (*Robot* 2021). Robots are used for a wide variety of tasks, from assembling cars to vacuuming homes to dangerous material cleanup (*Robotics* 2021). Currently, most are used in industrial applications such as assembly, welding, packaging, or moving products. This is because it is typically faster, cheaper, or safer to use robots, especially for repetitive or dangerous tasks. Common manufacturers of industrial robots are Fanuc, ABB, and Universal Robots, where Fanuc and ABB focus on larger applications (welding, car assembly), and Universal Robots focuses on cobots, or robots that work closely with humans. Most of these robots are 6 degree of freedom (DOF) arms. Despite industrial settings being the largest market for robots, and 6 DOF robots being the most common, research is being pursued for other applications and types of robots as well.

Since the first robot was made in the early 1950s, research on robotic technologies has quickly grown (Roberts, 1999). This research involves a wide variety of areas, including military robots, humanoid robots, and soft robots. Up until now, the three main categories for humanoid robot application are high risk environments (mines, rescue, military, nuclear, industries, astronavigation), competitive (RoboCup, FIRA, DARAPA), and service (education, cognitive, home, medical, entertainment) (Saeedvand et al., 2019). However, it is important to note that the robots can fall in multiple categories. Examples are given in Table 1.1 below.

Table 1.1: Examples of Humanoid Robots

Category	Robot Name	Robot Weight (kg)	Cost (USD)	Research or For Sale
High Risk Environments	ASIMO	50	\$2,500,000 <i>(What happened to the Honda Robot? 2022)</i>	Research
	HUBO 2	45	\$400,000 <i>(The HUBO 2 humanoid robot is priced at \$400,000 2017)</i>	Research
Competitive	NAO	5.5kg <i>(Programmable Humanoid Robot NAO V6)</i>	\$10,500 <i>(Nao Robot V6)</i>	For sale
	Igus	15 <i>(Heney, 2018)</i>	\$6,200 <i>(Heney, 2018)</i>	For sale
Service	Poppy	3.5	\$10,637.59	For sale
	iCub	30 <i>(Maggiali et al., 2019)</i>	\$270,000 <i>(Saeedvand et al., 2019)</i>	Research

Humanoid robots are very different from those that are typically used in industry since industrial robots are typically 6 degrees of freedom (DOF) arms (shown in Figure 1.1). In addition to this difference, humanoid robots also face the unique challenge of walking on their own. Humanoid robots can be designed to walk on a variety of surfaces, such as flat ground, grass, or uneven environments. They can even be designed for more complicated movement tasks, such as climbing stairs, skiing, ice skating, and sloped environments (Saeedvand et al., 2019). Since humanoid robots are able to walk, they can access areas that may be difficult for other robots, especially ones on wheels. Therefore, it is also advantageous to give humanoid robots other abilities, such as the ability to grasp and hold objects. This allows them to access uneven areas and complete tasks, similarly to a human.

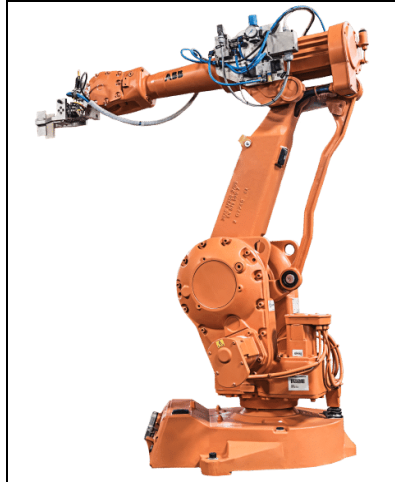


Figure 1.1: 6 Degree of Freedom Arm from ABB (reproduced as is from <https://roboticsys.com/robot-retrofit/>)

One limitation of humanoid robots is the high cost. As shown in Table 1 above, humanoid robots can cost anywhere from \$40 to \$2.5 million (*Ruko 6088 programmable robot; What happened to the Honda Robot? 2022*). Lower-end humanoid robots tend to lack the ability to walk, instead using wheels under the feet to move around. The steep price of walking humanoid robots makes it more difficult to expose students to this technology. Therefore, the objective of this research is to evaluate the Poppy humanoid robot and develop a low-cost version. Then we will give it the ability to walk on its own and grasp objects using actuated grippers.

1.1 Poppy

Work has been done to make humanoid robots more accessible to the general public. The Poppy project, a 3D printed open-source project, is a key example (Figure 1.1.1). Created by Matthieu Lapeyre, Pierre Rouanet, and Jonathan Grizou, Poppy is a 83cm tall robot that can be used for education, art, or science (Lapeyre et al., *Poppy humanoid: Advanced and easy to use open source humanoid robot*). Poppy is able to do a lot. It can replicate walking motions and dance, but the main goal of the project is to be a base so that others can add capabilities. The Poppy kit (which comes with all the necessary parts and assembly instructions) is available for purchase at €9,039.00 (\$10,637.59 USD) (*Discover the poppy open-source technology created at INRIA*), or can be made from scratch using the bill of materials and assembly instructions (<https://docs.poppy-project.org/en/>). The Bill of Materials for Poppy is in Appendix A.



Figure 1.1.1: Poppy humanoid robot (reproduced as is from <https://www.poppy-project.org/en/>)

In addition to the full Poppy humanoid robot, it is also possible to purchase the Poppy Torso. The Torso is the top half of the full Poppy. It is less expensive since it uses less motors and parts, and it can be sat on a desk (shown in Figure 1.1.2). The Poppy Torso kit (which comes with all the necessary parts and assembly instructions) is available for purchase at €5,300.00 (\$6,066.62 USD) (www.generationrobots.com/en/281-robot-poppy-torso), or can be made from scratch using the bill of materials and assembly instructions (docs.poppy-project.org/en/assembly-guides/poppy-torso/bom.html).



Figure 1.1.2: Poppy Torso humanoid robot (reproduced as is from <https://www.pinterest.com.au/pin/758715868446282498/>)

The creators of Poppy have published their research, including experiment results and design discussions. Their goal was to build a robotic platform allowing them to experiment in the real world (Lapeyre et al., 2014a). The project was initially a research project “to explore the role of embodiment and morphology properties on cognition and on the learning of sensori-motor tasks” (The story behind the Poppy Project).

The robot as a whole was designed with the ability for 3D printed parts to be quickly swapped out or added (Lapeyre et al., 2014a). Experiments such as testing the best thigh shape for stability utilize this capability. This also adds the possibility for future experimentation, such as trying to give Poppy the ability to jump or move more quickly. One interesting aspect of Poppy’s design is its proportionality to human dimensions. This allows for a human-like walking gait (Lapeyre et al., 2013) making the project and research relevant to humans.

Poppy’s legs were the parts that required the most design time. The legs are made up of the hip, thigh, and feet. As mentioned previously, Poppy was designed with similar proportions to a human. For example, the designers of Poppy put its thigh at an inwards incline of 6° , matching a human thigh (Lapeyre et al., 2013). This increases stability when Poppy is walking. By keeping the robot’s base towards the center of mass, it increases the time it takes for Poppy to fall over when standing on just one foot, as shown in Figure 1.1.3 below (Lapeyre et al., 2013).

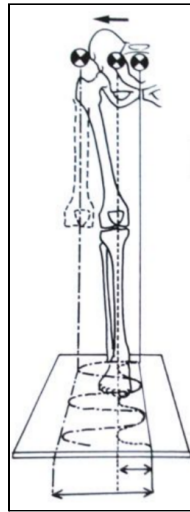


Figure 1.1.3: Thigh incline affects the center of mass (reproduced as is from Lapeyre et al., 2013)

The hip joints, the most powerful joints on Poppy, were created in a way to keep Poppy as stable as possible. Instead of having the hip be a ball joint, the designers placed the motors on the frontal plane as the left to right stability is greater than the rear to front stability (Lapeyre et al., 2013). By no longer using a ball joint, they are able to limit the mass on the back of the robot (Lapeyre et al., 2013).

Despite their small size, the feet also went through a large amount of design. Each foot has a single motor, weighing less than 200 grams (Lapeyre et al., 2013). Additionally, each foot has 8 FSF pressure sensors to provide accurate feedback of the state of the robot (Lapeyre et al.,

2013). The design of the legs, hips, and feet provide Poppy with the physical capability to balance and walk on its own.

Poppy is not the only child-sized humanoid robot. In fact, there are multiple humanoid robots around the same size as Poppy, in a range of materials and prices. A comparison of some common child-sized humanoid robots is shown in Table 1.1.1 below.

Table 1.1.1: Comparing specifications and hardware of child-sized humanoid robots (many 3D printed child-sized robots) (reproduced as is from Saeedvand et al., 2019)

Class	Robot	Size (cm)	Mass (kg)	Structure material	Actuators type	Processing unit	Sensors	Power	Cost (\$)
Child-sized humanoid robots	DARwIn-OP (Ha et al., 2011)	45.4	2.9	Aluminum, plastic	Electrical	Atom Z530 1.6 GHz CPU (32 bit) on-board 4 GB flash SSD	Camera, IMU	Li-Po 11.1 V 1 Ah	9600 (2019)
	ROBOTIS-OP3 ('ROBOTIS OP2-OP3')	51.04	3.5	Aluminum	Electrical	Intel NUC i3, 8 GB DDR4-120 GB, ARM cortex-M7	Camera, IMU	-	11 000 (2019)
	NAO (Gouaillier et al., 2009)	57	5.2	ABS	Electrical	ATOM Z530 1.6 GHz CPU, 1 GB RAM2, 2 GB Flash memory, 8 GB Micro SDHC	Camera, IMU, force, sensitive resistors, sonar, rotary encoders, tactile	Li-ion 21.6 V 2.25 Ah	9000
	Poppy (Lapeyre et al., 2014)	83	3.5	3D print	Electrical	Raspberry Pi	Stereo micros, force IMU	-	10 250 (2013)
	Igus (Allgeuer et al., 2015)	92	6.6	3D print	Electrical	Intel i7, 2.4 GHz CPU 4GB RAM, 120GB SSD	Camera, IMU, encoders	Li-Po 14.8 V 3.8 Ah	-
	iCub (Metta et al., 2010)	104	22	3D print	Electrical	Microcontroller based Freescale 56F807 chip	Cameras, IMU, force, torque, microphone	Li-ion -	270 000 (2016)
	ARC (Saeedvand et al., 2018)	54	2.9	3D print	Electrical	Intel i5, 1.8 GHz CPU 32GB RAM	-	Li-Po 11.1 V 2.3 Ah	-
	Surena mini (Nikkah et al., 2017)	53.4	3.3	3D print	Electrical	Intel coré m5, 1.1 GHz CPU, 4GB DDR3, 64 GB eMMC	Camera, microphone, force, IMU, IR, touche, encoder	Li-Po 14.8 V 2.3 Ah	8000 (2017)

As shown in the table above, Poppy is fairly expensive at \$10,250 USD. The majority of Poppy's price comes from the motors, with motors costing nearly \$7,000 USD. Poppy's motors and motor cost are shown in Table 1.1.2 below. The motor locations are shown in Figure 1.1.4 below.

Table 1.1.2: Poppy motor prices

Motor	Number of Units	Cost per Unit	Total Cost
AX-12A	2	\$58.43	\$6,829.68
MX-28AT	19	\$310.56	
MX-64AT	4	\$406.13	

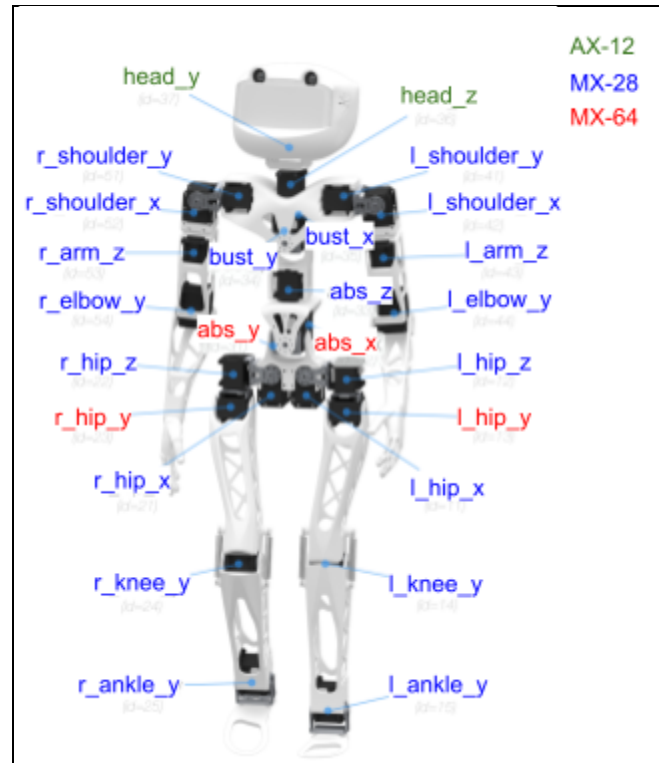


Figure 1.1.4: Poppy’s 25 motors (reproduced as is from Lapeyre, 2014b)

These 25 motors are divided into several groups: head, torso, arms, and legs. The head group is the smallest, consisting of 2 AX-12 neck motors. These motors provide the head with the ability to swivel and nod. The arms each contain 4 joints, each powered by an MX-28 motor. This gives Poppy’s arms 3 degrees of freedom at the shoulder and one at the elbow. The torso contains 5 motors. One MX-28 rotates the torso. Above and below this point is a double rotation section consisting of coaxial motors giving 2 degrees of freedom at the same point. The upper double rotation (“bust” in the image above) uses MX-28’s while the lower double rotation (“abs”) uses more powerful MX-64 motors to support the robot’s weight. Finally, each leg consists of 5 joints: 3 hip joints (2 MX-28 and 1 MX-64), an MX-28 knee joint with spring support, and an MX-28 ankle.

Poppy is undergoing continuous improvement. One area for future work highlighted by the designers is giving Poppy the ability to grasp (Lapeyre et al., 2013). This would be done by actuating Poppy’s hands in a compliant way (Lapeyre et al., 2013). This is in contrast to the hands Poppy was designed with, which are solid, non-compliant models in the shape of a human hand.

1.2 Poppy Redesigns

Many teams have worked to make a less expensive Poppy. As mentioned previously, the total cost for Poppy's 26 motors is nearly \$7,000, making them by far the most expensive component of Poppy. Due to this high price, most of the Poppy remakes have looked into using less expensive motors. This often meant that the motors needed to be modified in order to contain the needed sensors. One Poppy remake used SpringRC SR-518 motors (\$53), which have 2Nm of stall torque and use RS485 bus connections (Popov et al., 2017). To fit the new motors, the researchers modified the mounting places on the 3D printed parts.. These motors were successful in most joints, but the researchers recommend using stronger motors in the hip joints (Popov et al., 2017). Additionally, these motors are currently unavailable for purchase. Prasanna, who worked on another Poppy remake, used modified Hitec servos (HS-7954SH, \$120) as they are cheaper than the original Dynamixel motors (Prasanna & Ashok, 2021). Their modifications included removing all the original electronics and adding in a motor driver (VNH5180A-E), a magnetic encoder (Austria Microsystems AS5145), and an arduino microcontroller. Unfortunately, these motors are currently unavailable for purchase. Iglesias used the same motors and setup as the Prasanna remake. Table 1.2.1 compares the original Dynamixel MX-28AT motors to the Hitec HS-7954SH used by Prasanna and Iglesias (Iglesias et al., 2016).

Table 1.2.1: Comparing Dynamixel motor with Hitec motor (reproduced as is from Iglesias et al., 2016)

	Dynamixel MX-28AT	Hitec HS-7954SH
Operating voltage	12V	7.4V
Stall Torque	2.5 Nm	2.84 Nm
No-load speed	55 rpm	83 rpm
Weight	72 g	66 g
Resolution	0.088 °	0.088 °
Operating Angle	0-360 °	0-360 °
Max Curren	1.4 A	2.6 A
Price	240 €	105 €

Due to the different motors used, most of the Poppy remakes needed to redesign Poppy in order to fit the new actuators. Some of these modifications were slight, such as modifying the motor horn hole pattern, but Iglesias's team had to widen the thigh and shin to fit the larger actuators, which required more time (Iglesias et al., 2016). This is because the Hitec actuators are 1.6 x 0.8 x 1.5 inches (*HS-7954SH High Voltage Ultra Torque Servo*), while the Dynamixel actuators are 1.4 x 1.99 x 1.4 inches (*Dynamixel MX-28AT*). Additionally, some of these remakes modified the parts just so that they are easier to 3D print. Iglesias made the thigh straight so that it is easier to print on an FDM printer (Iglesias et al., 2016). They also split both the thighs and shins into two parts for easier assembly and reparability. The modified thighs can be seen in

Figure 1.2.1a and 1.2.1b below. These larger, straight legs can be seen in Figure 1.2.2 below. These modified legs are 434.1mm tall and 61.7mm wide, while the original Poppy legs are about 401.2mm tall and 52.7mm wide (Iglesias et al., 2016; poppy cad).



Figure 1.2.1a: The first version of the modified thigh compared to the original Poppy thigh (reproduced as is from Iglesias et al., 2016)



Figure 1.2.1b: The first version of the modified thigh compared to the final thigh design (reproduced as is from Iglesias et al., 2016)

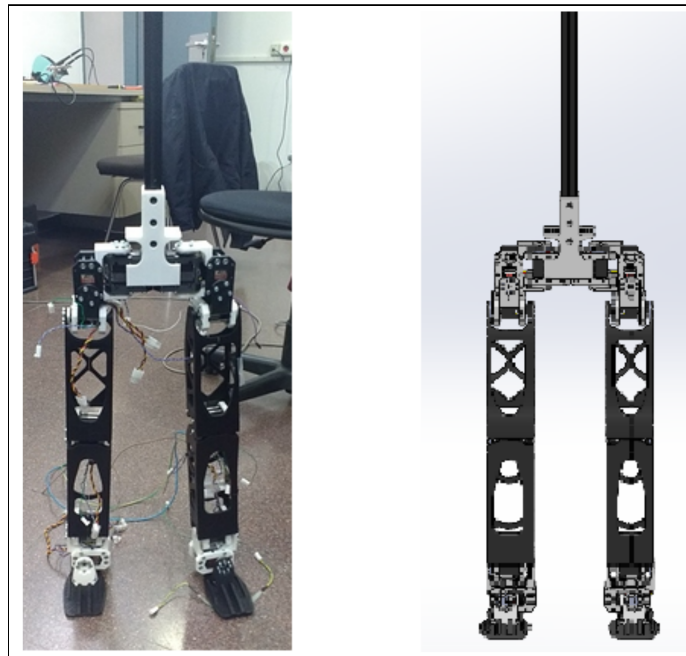


Figure 1.2.2: Poppy remake legs (reproduced as is from Iglesias et al., 2016)

Despite all the research that has gone into them, none of these remakes give Poppy the ability to walk unassisted. Therefore, giving Poppy the ability to walk on its own is another objective of this project. This could involve a mixture of hardware changes - such as leg and foot variations - or software changes to add new balancing algorithms.

1.3 Objectives

Therefore, using the research we have found and the experiments and prototyping we plan to do, our objectives are to:

- Build a full humanoid robot based on the Poppy Project
- Select alternative components to reduce the total cost
- Give Poppy the ability to walk on its own
- Give Poppy the ability to grasp and manipulate objects with actuated hands

1.4 Section Overview

The rest of the paper is laid out as follows:

- Chapter 2 discusses the literature review conducted on both humanoid gait and hand actuation methods.
- Chapter 3 discusses the project plan and an overview of the methods and processes used to complete tasks throughout the project.
- Chapter 4 discusses the specific design changes made from the Poppy project to Koalby with primarily a mechanical focus, along with a brief section on electrical changes.
- Chapter 5 discusses the manufacturing and assembly processes used throughout the project.
- Chapter 6 discusses the electronic side of the project including both power and control.
- Chapter 7 discusses the code behind the project including both the firmware and the high-level code.
- Chapter 8 discusses the various testing methods used throughout the project to evaluate the system.
- Chapter 9 discusses the final state of all the various goals and challenges that were encountered along the way.
- Chapter 10 discusses the final state of the project as a whole, the future work that can be done to further the project, and the experience group members had during the project.

2 Literature Review

2.1 Humanoid Gait

While Poppy is currently unable to walk on its own, the math and models for biped robots have been developed in other humanoid robot projects. The current standard for biped gaits is based on the linear inverted pendulum model (Iverach-Brereton et al., 2014). As given by the name, this gait models the robot as an inverted pendulum. The robot's support foot (the foot on the ground) is the fulcrum, the robot's support leg (connected to the support foot) is the rod, and the robot's upper body is the mass, as shown in Figure 2.1.1 below.

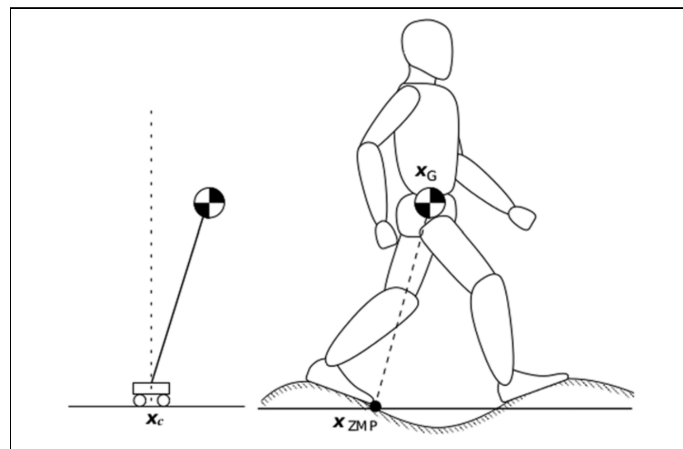


Figure 2.1.1: Inverted pendulum model for the human gait (reproduced as is from Iverach-Brereton et al., 2014)

The zero moment point, ZMP, at the end of the pendulum can be manipulated by varying the length and angle of the rod. This controls the robot's center of mass (CoM), which controls if the robot is able to balance as it walks or not (Iverach-Brereton et al., 2014). In the pendulum model, there are two gait phases: the single support phase (SSP) and the double support phase (DSP). DSP is when the robot is statically stable with both feet on the ground, and the CoM is above the support polygon formed by the feet (Iverach-Brereton et al., 2014). SSP is essentially when the robot is walking; the front foot is the support leg, and the robot pivots about the ZMP, causing it to fall forwards in a walking motion (Iverach-Brereton et al., 2014). Many humanoid robots use gyroscopes to measure a robot's angular velocity along the X, Y, and Z axis, which allows them to detect when a robot has transitioned from the SSP to the DSP (Iverach-Brereton et al., 2014).

Additionally, it is possible to give robots the ability to walk with assistance. One example is pushing a shopping cart, where the shopping cart helps to balance the robot. As mentioned previously, Poppy has the hardware (joints, surface area, center of mass) needed to walk on its own. What is currently missing from Poppy is the balancing algorithms. Our goal is to

implement a balancing algorithm for Poppy to walk. If this proves to be too difficult, then we will implement an assistance tool, such as a shopping cart.

2.2 Actuated Hands

In addition to walking, research was conducted on humanoid robots with the ability to grasp. One such robot is Epi, an open-source humanoid robot. This humanoid robot is a bit larger than Poppy at 101cm tall and weighing 12kg (while Poppy is 83cm tall and weighs 3.5kg) (Johansson et al., 2020; *Poppy humanoid: Advanced and easy to use open source humanoid robot*). Despite the size difference, Epi is also a 3D printed robot with similar proportions to a human (Johansson et al., 2020). Epi's hands each have 4 moveable fingers and a stationary thumb, somewhat similar to a human, seen in Figure 2.2.1 below.

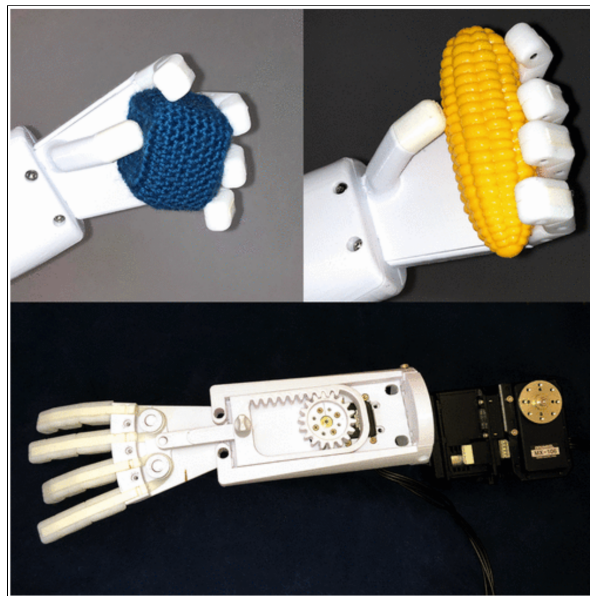


Figure 2.2.1: Epi 5 figure hand design (reproduced as is from Johansson et al., 2020)

Each hand is actuated with a single Dynamixel MX-106 servo motor (Johansson et al., 2020). This allows the hands to lift objects up to 1kg in weight. The fingers are mounted at angles to each other so that when closed, the fingertips all meet at one point, which produces a grasp movement (Johansson et al., 2020). The fingers are actuated using 3D printed rubber tendons, as shown in the image above (Johansson et al., 2020). Epi has one such robotic hand design that may be integrated into Poppy.

NICO is another open-source 3D printed humanoid robot project that explored grasping, though it did so with three-fingered hands (Kerzel et al., 2017). All three fingers were actuated with 3-segments, and the symmetry in the design means that the hand could be duplicated on the right or left arm without affecting its ability or the aesthetic appearance in any way. The hand can be seen in Figure 2.2.2 below. In total, this resulted in 10 degrees of freedom with only two actuators, and included wrist rotation. They were also equipped with force sensors at the

“thumb” finger tip for tactile sensing to assist with the gripping ability. These hands were provided through a company Seed Robotics which has since discontinued the product (*RH2D Advanced manipulator*).

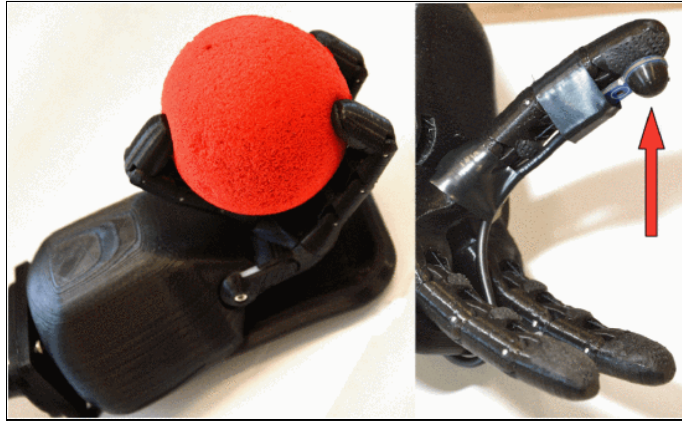


Figure 2.2.2: NICO 3 finger hand design (reproduced as is from Kerzel et al., 2017)

Vizzy is yet another humanoid robot project focused on the ability of the robot to grasp objects. Vizzy’s fingers are actuated using 3 motors and strings. Each motor pulls one or more strings that are attached to the tips of the fingers. The thumb and pointer finger are each actuated with one motor; the last two fingers are actuated by the 3rd motor (Moreno et al., 2015). One of the largest differences between the Vizzy project and NICO, the previous projects mentioned, is that Vizzy’s multi-motor arrangement allows for the hand to perform a few different types of grasping techniques, including three types of power grasps (cylindrical, spherical and hook) and one type of precision grip (tip-to-tip) (Moreno et al., 2015). The CAD of the hand and pictures of the grasping ability of the hand are pictured in Figure 2.2.3 below.

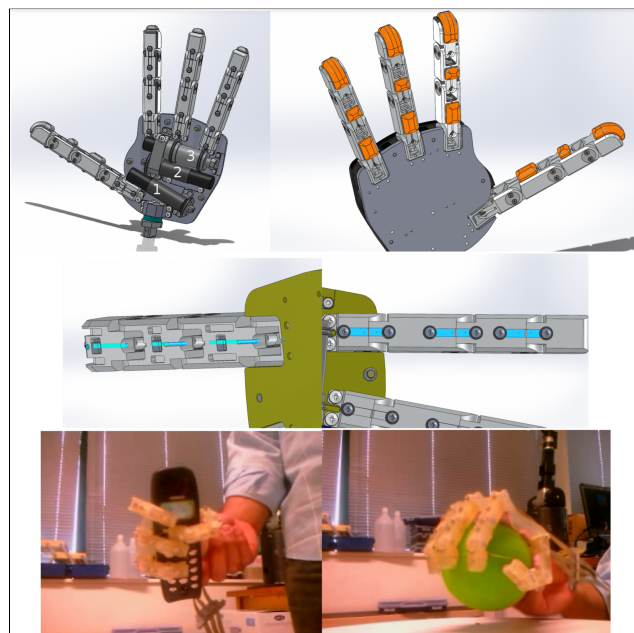


Figure 2.2.3: Vizzy 4 finger robotic hand (reproduced as is from Moreno et al., 2015)

Research was also done into previous robotic hand projects at Worcester Polytechnic Institute. All the robotic hands developed at Worcester Polytechnic Institute are closer to the size of a human hand, much larger than what Poppy would have. Despite this, these projects can still help to develop our own, smaller, robotic hand. One of the projects done was named Accurate Prosthetic Hand. The hand was designed to be analogous to parts of a human hand, such as bones and joint caps, muscular tendon simulant, the thumb, and the wrist. The fingers move just like in a human hand, with the use of tendons pulling the fingers. The hand design can be seen in Figure 2.2.4 below. The complete hand has 9 DOFs, uses 9 motors, and costs \$350 (Larrier et al., 2017).

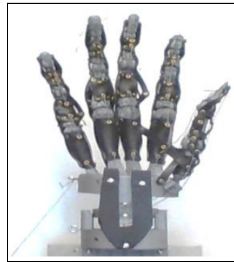


Figure 2.2.4: Prosthetic hand from 2017 MQP (reproduced as is from Larrier et al., 2017)

Another robotic hand project at WPI was IRIS Hand. IRIS Hand works very similarly to the Accurate Prosthetic Hand; it uses tendons to move each of the fingers. The hand design can be seen in Figure 2.2.5 below. The complete hand has 6 DOFs, uses 6 motors, and costs \$1,800 (Casley et al., 2014).

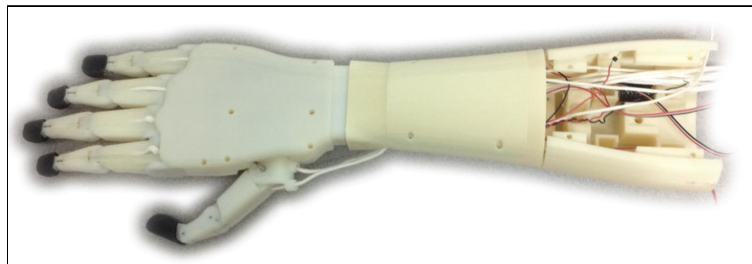


Figure 2.2.5: Prosthetic hand from 2014 MQP (reproduced as is from Casley et al., 2014)

3 Project Plan and Methodology

This chapter discusses the process of adapting, manufacturing, and building the team's version of the Poppy robot. The modified robot is called Koalby, and it uses a different motor to reduce cost, necessitating changes to hardware, electrical systems, and software.

3.1 Motor Selection

To decrease system cost, Koalby uses a less expensive smart motor in place of the Dynamixel MX-28 (MX-28). This alternative motor, the HerkuleX DRS-0201 (HerkuleX), is similar in size and power output to the MX-28 but uses a different 4-wire bus system and form factor. The motor comparison is shown in Table 3.1.1 below.

Table 3.1.1: Comparing the MX-28 to the HerkuleX

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

With 17 MX-28 motors on the full system, the total cost savings from this change is ~\$2210. This is the largest difference between Koalby and the base Poppy Project, as it necessitates a series of further hardware and electronic changes to install and control these motors. Poppy uses 3 types of Dynamixel motors: the large MX-64, medium MX-28, and small AX-12, all of which communicate on a 3 wire bus, which is not compatible with the 4-wire HerkuleX motors.

Table 3.1.2: HerkuleX Library Functions

torqueON(int servoid)	set the motor torque to ON
torqueOFF(int servoid)	set the motor torque to OFF
moveAll(int servoid, int Goal, int iLed)	prepare the motors to execute the movement in the same time to position (same start time – end time) – unison movement
moveAllAngle(int servoid, float angle, int iLed)	prepare the motors to execute the movement in the same time to angle (same start time – end time) – unison movement
moveSpeedAll(int servoid, int Goal, int iLed)	prepare the motors to execute the movement in continuous rotation with same start time and end time – unison movement
actionAll(int pTime)	execute the unison movement
moveSpeedOne(int servoid, int Goal, int pTime, int iLed)	motor continuous rotation with a specified speed
moveOne(int servoid, int Goal, int pTime, int iLed)	move one motor to an absolute position
moveOneAngle(int servoid, float angle, int pTime, int iLed)	move one motor to an angle
getPosition(int servoid)	get the motor position
getAngle(int servoid)	get the position in degrees
getSpeed(int servoid)	get the motor speed (continuous rotation)
reboot(int servoid)	reboot the motor
setLed(int servoid, int valueLed)	set the led color

With their lower price, HerkuleX motors have correspondingly lower capabilities. Control of these motors is effectively limited to setting the target position, getting the current position, and enabling or disabling torque as shown in the library documentation above (Table 3.1.2). This is more limited than the Dynamixel motors, which feature the ability to set torque and velocity limits, allowing for features like compliant mode, where the motor holds its position with low torque so that it can be repositioned by the user. Poppy utilizes some of these advanced features, and Koalby required additional programming to account for the more limited sensing and control the HerkuleX motors provide.

3.2 Redesigning for HerkuleX

The HerkuleX motors have a different form factor than the MX-28s that the Poppy robot was originally designed for (specific dimensions shown in Table 3.1.2 above). Numerous small redesigns (see section 4.1) were required to adapt the existing Poppy CAD files to utilize the new motors.

The team started by redesigning the parts in the robot's torso and arms. This allowed preliminary testing to begin concurrently with the redesign of the pelvis and legs. All of the joints that used MX-28 motors (see section 1.1) had to be redesigned. The AX-12 in the head of the robot and the MX-64 joints in the lower torso and hips of the robot were both kept in the

design as there were no viable alternatives which would decrease costs enough to justify the engineering effort required to modify the robot to utilize them.

The HerkuleX motors are smaller than the MX-28's in all dimensions (see Table 3.1.1 above). Therefore, swapping to these motors did not require a major redesign, as HerkuleX motors could be inserted with adaptors that allow them to use the hole patterns originally designed for the larger MX-28. There also were certain locations in the robot where two motors were linked together to create two axes of rotation. The team designed a few new parts to allow the HerkuleX motors to be linked together in the same fashion, but again did not need to majorly redesign the robot as this new assembly could be installed in the same way as the MX-28 without any changes to the main part.

3.3 Printing the Structure

Poppy is an open source 3D printed robot, and the robot the team was creating was designed to be manufactured the same way. The team utilized two different methods of 3D printing, Fused Deposition Modeling (FDM) and Digital Light Processing (DLP). FDM printers feed continuous plastic filament from a spool through a moving and heated toolhead to lay the plastic in stacked layers. DLP printers use an LCD screen to cure layers of liquid resin onto a print bed. Using an LCD screen in place of the moving toolhead of an FDM printer allows for much higher resolution parts to be created, and allows for an entire layer to be created at the same time.

3.3.1 FDM Printing

For initial testing, the team printed using polylactic acid (PLA) filament on FDM 3D printers. After the PLA parts proved too fragile, the team switched to using polyethylene terephthalate glycol (PETG) filament. PETG is a thermoplastic polyester that is less rigid than PLA, but has a higher tensile strength (7700 psi) (*PETG*) than PLA (7250 psi) (Langnau).

The team began by printing the left arm subassembly. We printed the hand, forearm, upper arm, shoulder, and arm connector. Figure 3.3.1 shows an example of one of these parts, the upper arm, being printed. However, even when printing in PETG it was clear that FDM printing was not suited for the complex geometry of the parts. Prints would repeatedly fail and even when they did succeed, they would have many small imperfections. While the arm was possible to print using FDM techniques, the torso and legs seemed unlikely to succeed, and each failed print was only a waste of time.

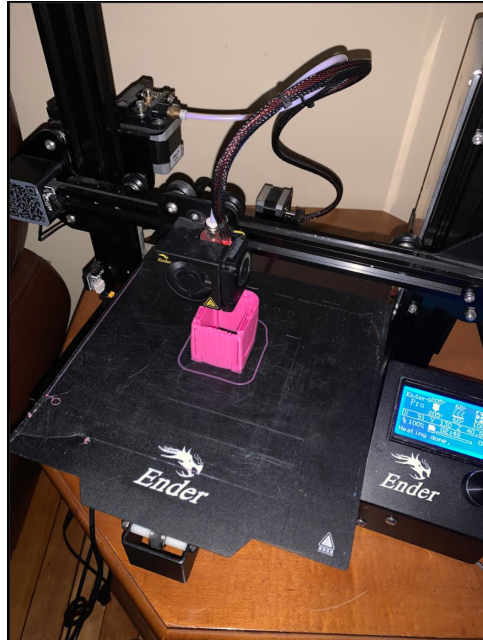


Figure 3.3.1: Photograph showing the upper arm being 3D printed in PLA filament

3.3.2 Resin Printing

Due to the challenges of FDM printing these parts, the team decided to print the parts using a resin printer as the Poppy Project originally designed them for. For this, the team utilized Elegoo Mars 2 Pro and Elegoo Saturn DLP printers with print areas of 5 in x 3 in x 6 in and 7.55 in x 4.72 in x 7.87 in respectively. The Mars 2 Pro printer is able to print most parts of the Poppy robot; however, several parts - specifically the torso and the shins - are too large for the relatively small print volume, which necessitated the purchase of the larger Saturn printer. The team's printer set up can be seen in Figure 3.3.2 below.

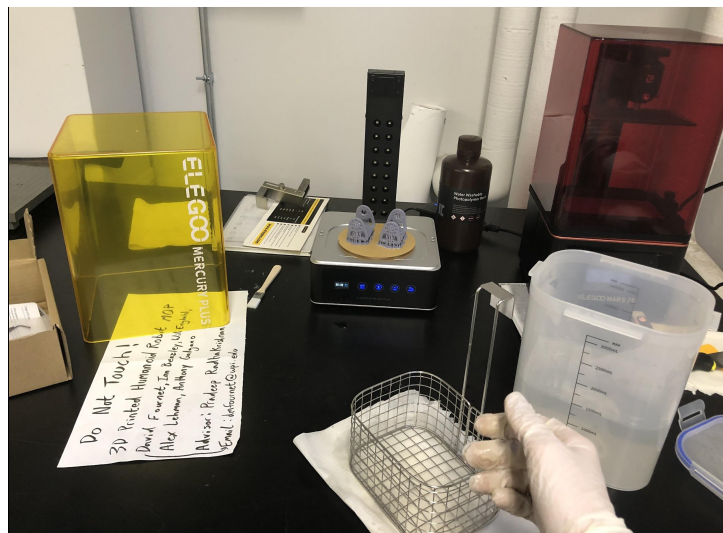


Figure 3.3.2: Elegoo Mars resin printer (right) and washing/curing machine (left)

3.4 Using the HerkuleX motors

HerkuleX motors can be controlled directly from the Arduino without further electronics (see section 7.1). Their 4 wire bus has separate unidirectional communication lines which work with Arduino input and output pins, while the shared data wire of the Dynamixel requires additional hardware. Each motor was tested individually to verify its functionality and configure them with a hexadecimal ID used to communicate with the specific motor on the shared bus.

The standard Arduino HerkuleX library was used for this process, primarily the *set_ID* and *moveOneAngle* functions. The *set_ID* function sets the motor ID, a value from 0 to 253, that allows individual motors to be contacted through the shared bus communication lines. Each motor has a unique ID assigned before installation on the robot. The team also used the function *moveOneAngle*, which moves a motor to a given angle over a certain period of time. These controls are further explained in chapter X. This function takes motor ID as a parameter, so commands can be sent to individual motors.

3.5 Programming Koalby

As the robot was being assembled, the team also started to dedicate resources to writing code to control Koalby's mixture of HerkuleX and Dynamixel actuators. This included understanding how the code for the Poppy project had originally worked and whether it could be adapted for the Koalby project.

The team used an Arduino Mega 2560 to demonstrate initial motor functionality and verify successful mechanical design and electrical hardware selection at various levels of assembly en route to a fully functioning robot. This Arduino also provides an abstract interface for the Raspberry Pi to communicate with the separate types of smart motors. The Poppy Project did not use an Arduino because they had a custom electronic control board (<https://www.generationrobots.com/en/402420-carte-pixl.html>) that plugged directly into the GPIO pins on the Raspberry Pi. Koalby uses two separate motor bus systems, and the Arduino helps translate the Pi commands into messages on the correct bus.

At the same time, the team began to examine the original Poppy project code in hopes of being able to incorporate this code into the Koalby project, with only minor adjustments to account for motor changes and geometry adjustments. This proved to be more complicated than initially anticipated. The first issue they ran into was with inconsistencies between the latest documentation and the code itself. Tutorials created for the Poppy project and instructions on setting up the code (<https://docs.poppy-project.org/en/programming/python.html>) were not always consistent with how the code worked due to refactoring and renaming that had occurred after the tutorial had been last updated. This was something the team knew could be the case given the Poppy project is around 7 years old, but the issues still took significant debugging time.

After completing initial code setup and connecting key libraries, the team investigated how the Poppy code controls the robot. First, the team wanted to understand how humanoid actions, like moving the arms in planned motions, were implemented. This would be helpful for using the code itself, or applying the concepts to new code specifically written for Koalby if the structure of the project was not directly usable. The second goal was to determine how the specific motors were controlled through the code. Instead of sending commands to motors directly, the team needed to send all commands to the Arduino, which would relay those to the appropriate motors due to the combination of using both Dynamixel and HerkuleX motors.

Over the course of a few weeks of study, the team developed an understanding of how the Poppy project used a system it referred to as primitives to essentially schedule tasks to be done by the motors. Primitives allow actions to be combined for better control of the robot. A relevant example of this is the robot walking on uneven terrain. Sensor input would help the robot to maintain balance, and the primitive system would merge sensor feedback with the original stride of the robot to adjust for the terrain appropriately.

As the team continued to explore the code, the list of modifications needed in order for Koalby to work with the code kept growing. It started with an expectation that perhaps the robot class and a configuration file would need to be changed, and continued to grow with nearly every class the team examined. Additionally, there were many features the team did not need, such as the ability to work with a virtual robot, and the other versions of the Poppy project like the Poppy Torso and Poppy Ergo. These features added complexity, which would make the entire coding process take longer, or revolved around flexibility to work with other platforms when the humanoid robot was all that was needed for this project. Direct communication with the Dynamixel motors is integral to how the Poppy project code works. This direct communication is not necessary with Koalby's Arduino-message system since the Arduino can handle all incoming messages and send them forwards to the appropriate motors. These issues mean that adapting the Poppy code would take more effort than developing new code from scratch based on Koalby's design. The concepts of the code were understood, but implementing the code itself would take time, so the team instead pivoted to writing their own code specifically for Koalby. This code would include basic motor communication from the Pi to the Arduino, kinematic control of the robot, and a primitive system similar to that of the Poppy project. Many features, such as kinematics, will be inspired by their Poppy implementation. Overall, it will be much more simplistic than the Poppy code and include room to grow into future projects. This code development is detailed in section 7.

3.6 Gantt Chart Details

The project can largely be divided into two phases: Preliminary Research and Part Validation (August-November), and Assembly and Systems Testing (October-May). The first phase focused on understanding the Poppy Project, acquiring components, and testing modifications to the project such as FDM printing or HerkuleX motors. In the second phase, the team built the full humanoid robot, tested it, and made a number of improvements for our end of

3.7 Term Progress

The Gantt charts above show term progress. The team spent A-term familiarizing itself with the Poppy Project, purchasing parts, and testing individual components and small subystems, ending by starting work on the complete HerkuleX torso. The team spent B-term building the first version of Koalby. C-term was used for completing assembly, working on code, and fixing numerous issues that were found through full robot testing. In D-term, the team readied Koalby for demonstrations and completed documentation for future groups to take over the project.

3.7.1 A Term - Research and Purchasing

Once the team decided on our goal of reducing the costs of building Poppy, they saw that the easiest way to do so is to use less expensive motors. Therefore, the first step in A Term was finding less expensive motors that are comparable to the Dynamixel's torque and form factor. Once the team found motors, they ordered some to build one of Poppy's arms, this way the team could test that the motors would work as a suitable replacement. This required the team to also modify the arm so that the new motors could fit.

Research was also done on our other goal, making Koalby battery operated. The team looked specifically at what voltage and current was needed to operate the motors, and therefore determined the battery requirements.

While this was taking place, the team also did research on other existing humanoid robots to see what can be applied to Koalby. Interest was taken specifically in how humanoid robots walk and in actuated grasping hands. The results of this research can be seen in section 2.

This term ended with the beginning of robot assembly and printing. The team tested FDM printing (section 3.3.1) alongside Resin printing (section 3.3.2) and assembled prototype arms with both techniques. Testing was done on these arm prototypes (section 8.1), which determined the new Herkulex motors are adequate replacements for the original MX-28's. Once this was confirmed, the rest of the robot could be redesigned for HerkuleX motors, starting with the torso. The team printed the torso parts as the redesign was completed.

3.7.2.B Term - Initial Assembly

With the redesign for HerkuleX motors in progress, the team began to assemble the newly designed torso while continuing to develop the modified legs. During this term, the team selected batteries for the robot (section 6.1) as the leg redesign added space to install these batteries in the shins (section 4.2.1).

With the torso assembly concluded, the team began programming the robot (section 3.5). Here the team began to develop Arduino firmware to control the HerkuleX motors. It was at this point that the team realized that the existing Poppy code could not be used, and began to rewrite it for Koalby. As the torso programming continued, the team also worked on completing the leg redesign and printing the parts to complete assembly in the beginning of C term.

3.7.3 C Term - Code and Revisions

While C term began with the complete robot being assembled for the first time, assembly was not the primary focus of this term. Development of the code and the routine testing (section 8.2) that went along with this progress was primarily how time was spent throughout the term. This was the term where all the final code for the robot was developed including the primitive manager and the record/replay functionality (section 7). The goal of creating graspable hands was also addressed this term with the creation and assembly of a simple 1-dof hand (section 4.2.2).

3.7.4 D Term - Finalizing Year's Progress

D term was the least developmental term of the project with a focus on finalizing the polished demonstrable robot and documenting all progress for future work to pick up where this project left off. This term was still key in testing the robot through events like the WPI TouchTomorrow event (section 8.3) which allowed for battery life to be thoroughly tested. Additionally, documentation like the assembly instructions (Appendix D) were created specifically to allow for the project to be recreated and continued both by future WPI teams who are already planning to continue the work, as well as any other groups who would be interested in continuing to develop this platform.

4 Design Changes

This MQP is based on the open source Poppy Project; however, many changes were made to the project to achieve the team's goals, as well as when encountering issues in getting the project to perform as expected. These changes primarily occurred in the areas of design changes and code changes.

4.1 Design Modifications for HerkuleX

One of the largest goals of the project was to reduce the original ~\$7,000 cost of assembly the Poppy Project required. To accomplish this goal, the team decided to switch away from the MX-28 motors that the Poppy Project uses and utilize the cheaper HerkuleX DRS-0201. These motors do not have the same form factor as the original motors, which necessitated a number of design changes to use these motors. The original and new costs for the robot are shown in Appendix B.

4.1.1 Motor Adapter

The first change that was made to the design revolved around the motor's form factor. The HerkuleX motors are significantly thinner and slightly smaller than the MX-28 motors in most other dimensions (see Table 3.1.1). 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. To solve this issue, the team created a motor adapter that mounts to the HerkuleX motors and allows the motor to be mounted to the robot using the original MX-28 mounting points. This adapter piece is shown in gray in Figure 4.1.2 and is attached to both sides of the motor. Creating this motor adapter significantly reduced the time it would have taken to modify the CAD since this sub-assembly could be used wherever a MX-28 had been used in the original project.

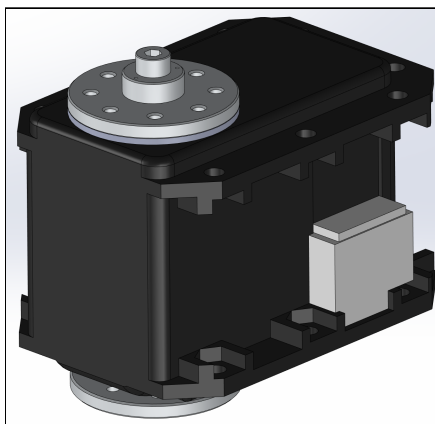


Figure 4.1.1: Two Horn Dynamixel motor

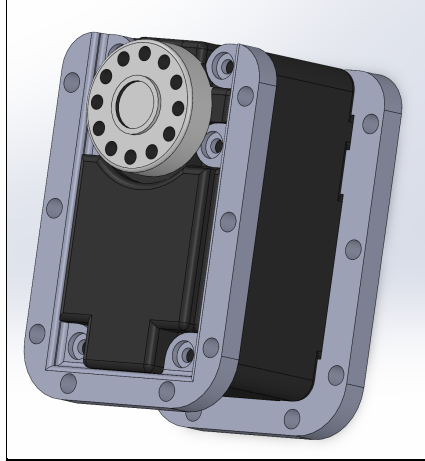


Figure 4.1.2: HerkuleX Motor CAD with the motor adapter prints attached

This was not the only change needed to substitute the HerkuleX motors. MX-28 motors utilize a two horn design with one horn on either side of the motor as shown in Figure 4.1.1. HerkuleX motors only came with one horn. Additionally, the total width from horn to horn on the MX-28 motors (41mm) was much larger than the HerkuleX motor (~33.5mm). The team developed a 7.4mm spacer that connects to the other side of the HerkuleX motor to solve both of these issues. This cylindrical piece can be seen in Figure 4.1.3 and the comparison of its spacing to that of a MX-28 can be seen in Figure 4.1.4. The spacer has a hole pattern matching that of a HerkuleX servo horn to support the motor on both sides.

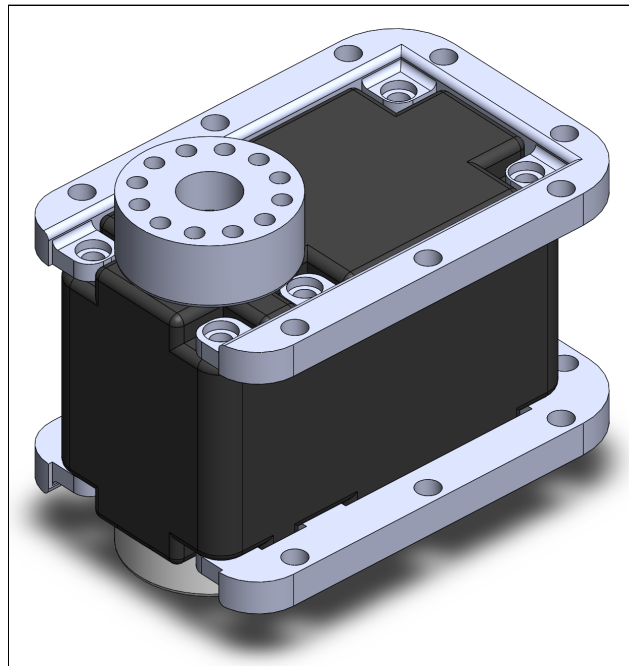


Figure 4.1.3: HerkuleX motor with spacer

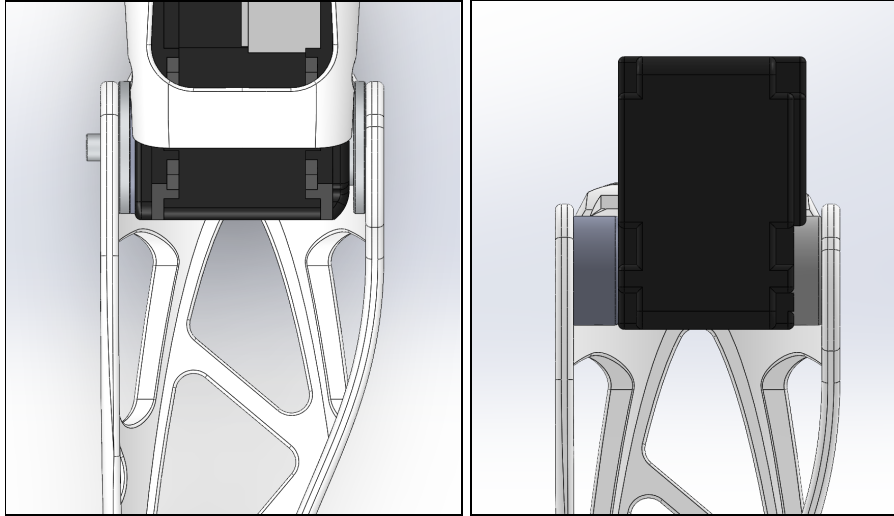


Figure 4.1.4: Comparison of the Spacing of the original MX-28 implementation (left) and the modified HerkuleX implementation of the spacer (right)

4.1.2 Motor Double Rotation

HerkuleX motors can be installed in place of 19 MX-28 motors with the simple addition of the adapter and spacer discussed above. However, there are two locations in the torso where two motors are linked together to allow for rotation on two separate axes. Figure 4.1.5 shows this assembly and the version created for the HerkuleX motors to accomplish the same functionality. The middle linking piece length was created to match the positions of the HerkuleX motor horns with the original MX-28 positions. The piece where the spacer is mounted is positioned to give a 2.5mm buffer for a screw to be mounted without scraping the motor itself.

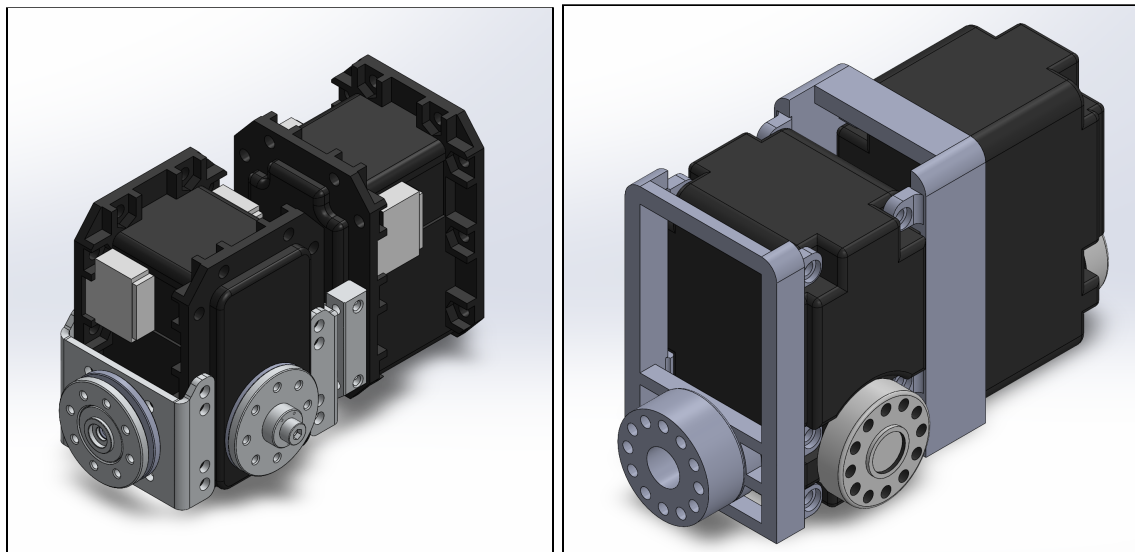


Figure 4.1.5: Comparison of original Dynamixel double rotation assembly (left) and the HerkuleX version (right)

4.1.3 Arm

Modifying the individual parts of the robot required much less work due to the motor adaptor prints. However, the hole patterns for the servo horns had to be modified to match the hole pattern of the HerkuleX motors. All of these hole patterns had to be converted from the 8 hole design of the MX-28 motors to a 12 hole design. This design change can be seen in Figure 4.1.6. This was the one major change needed to allow a Herkulex motor with an adaptor to replace an MX-28.

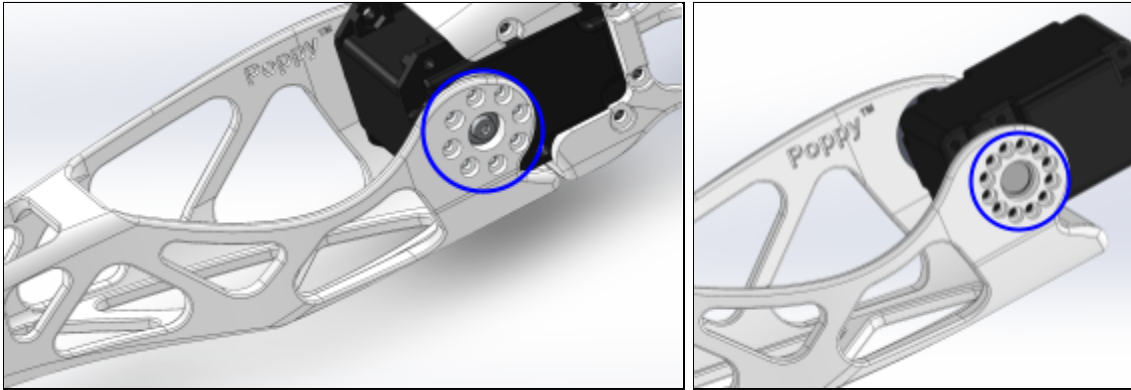


Figure 4.1.6: Comparison of original Dynamixel servo horn hole pattern (left) and the HerkuleX version (right)

The only other change that needed to be made to accommodate the motors was to check for interference. While the HerkuleX motors were almost uniformly smaller than their Dynamixel equivalent, there were a few locations where the HerkuleX motor could run into issues with interference with the arm itself. Figure 4.1.7 illustrates this with the arm connector piece that links the upper arm to the shoulder. In this print, the motor interferes with the print where it is circled in red. This was resolved by creating a cut-out wherever interference was happening. This interference check had to be performed for each HerkuleX motor added.

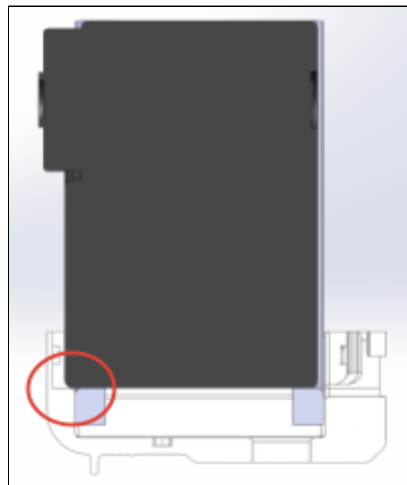


Figure 4.1.7: HerkuleX motor interference where the motor is clipping into the print

4.1.4 Torso

The torso was the next piece that the team redesigned. While it experienced many of the same small modifications needed for the arm, such as the changes in mounting hole patterns, it additionally required the removal of the handle from the Poppy torso (shown in blue in the image below). The original Poppy design included a handle built into the chest piece to allow for Poppy to be picked up (Figure 4.1.8). However this design did not fit on the resin print bed. This required the removal of the handle (Figure 4.1.9). With the handle removed, the chest piece was still too large for the print bed; however, it could be sliced such that a small edge at the bottom of the servo horn mounting holes was removed while the remaining torso fit on the print bed. This does not affect the robot's function, and the chest piece was printed in that configuration.

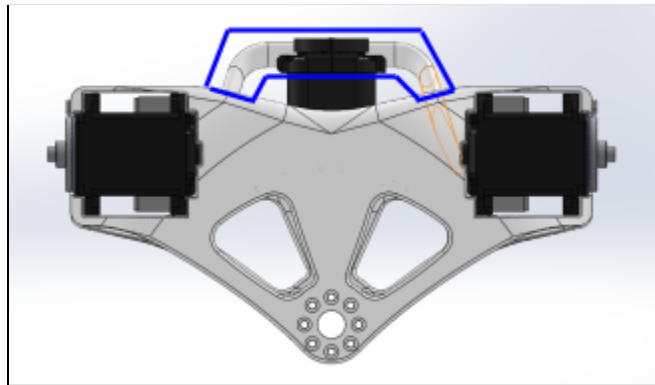


Figure 4.1.8: Original Poppy Chest Piece with Handle

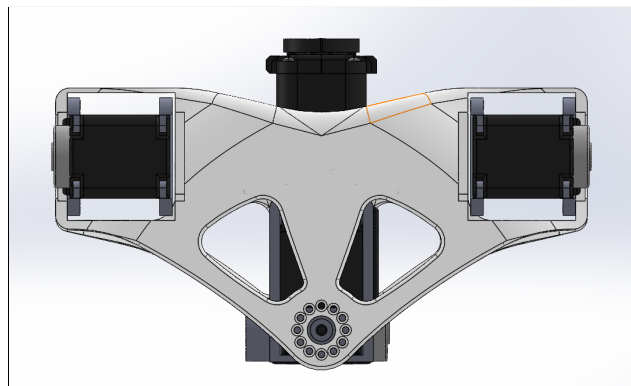


Figure 4.1.9: Koalby Redesign to fit on print bed

4.1.5 Lower Limbs

The lower limbs of Poppy are made up of the hip and pelvis assembly, thigh, and shin and foot assembly, all shown in Figure 4.1.10 below.

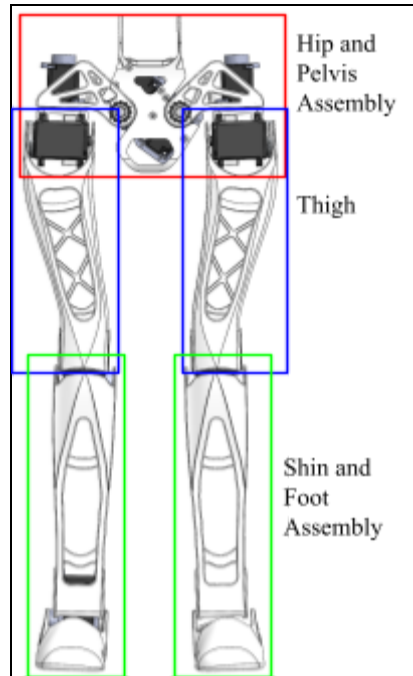


Figure 4.1.10: Poppy lower limb assembly.

Modifications to the pelvis assembly were minimal. As with the other parts, the motor horn attachment holes were modified to match the 12-hole motor horn on the HerkuleX motor. Additionally, the motor mounting holes for the hip joint needed to be moved in order to have the motor be at the same height. At the original position, the center of the servo horns had a height difference of 2.91mm. The HerkuleX motors at the original, incorrect position is shown in Figure 4.1.11 below.

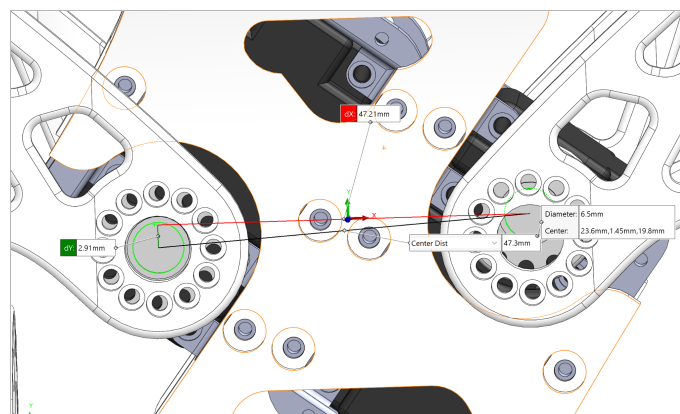


Figure 4.1.11: Incorrect location of the motor mounting holes.

The new locations of the motor mounting holes were determined by making the center of the servo horns concentric with the center of the circular cutout in the pelvis piece that they are attached to. The original pelvis assembly and the modified pelvis assembly are shown in Figure 4.1.12 below.

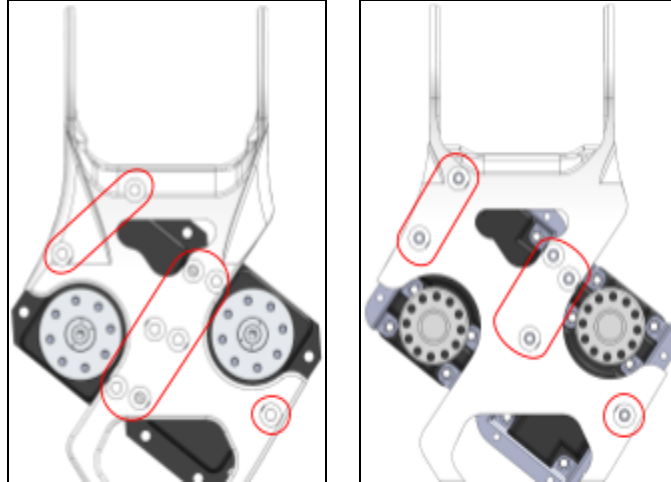


Figure 4.1.12: Original pelvis assembly (left) and modified pelvis assembly (right)

4.2 Mechanical Additions

In order to accomplish our goals of making Koalby battery powered and giving it actuating hands, larger additions to his design were required rather than the more modest redesigns required for motor substitution.

4.2.1 Shin and Foot

In addition to modifying the shin and foot servo horn holes to match the HerkuleX servo horn, modifications were made to fit a battery in the shin. One goal of the project is to make Koalby wireless with the use of two 7.4 V batteries (47.5mm x 26.5mm x 140mm batteries). The team decided that the best place to store the batteries was in the shin since it was the part with the largest amount of open space. Storing the batteries in the shin also helps to keep the center of mass lower to the ground. It was calculated that the thigh motors will be strong enough to lift the legs with the batteries in the shins (shown in Appendix F).

To fit the batteries, we made the shin 30mm taller and 19mm longer (front to back) as shown in figure 4.2.4. At its current height, the shin is too tall to fit on the DLP printer (shown in Figure 4.2.1 left), so it was split into two pieces that would be attached (shown in Figure 4.2.1 right).

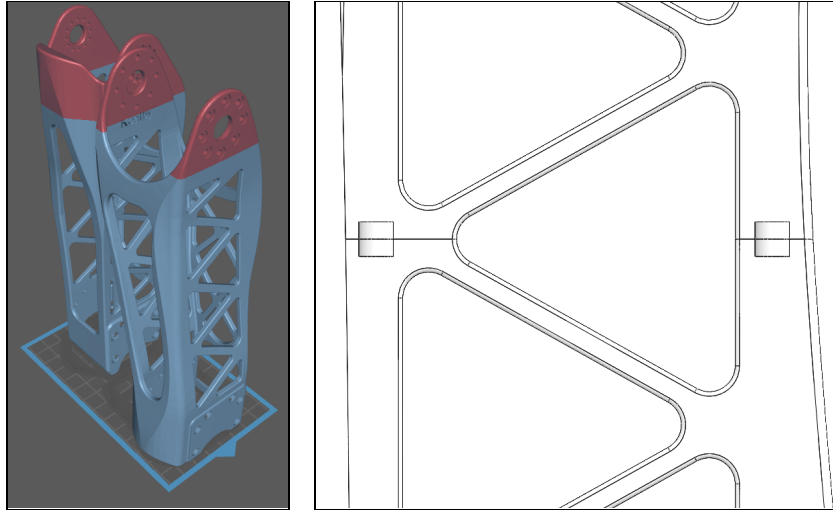


Figure 4.2.1: The shin was too tall to fit on the DLP printer (left) so it was split into two pieces (right).

A battery holding platform and upper stop were added to the shin to keep the battery from interfering with the motors (shown in Figure 4.2.2).

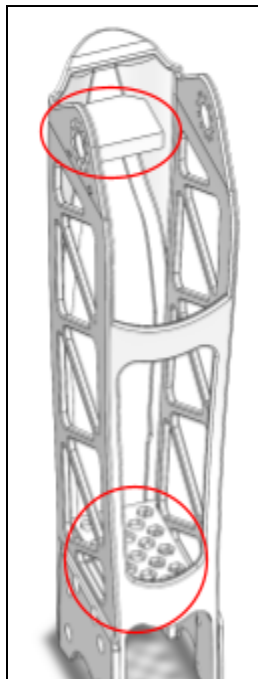


Figure 4.2.2: Battery platforms in the shin.

With the shin being 19mm longer front to back, the bottom of the shin interfered with the ankle joint's upward motion, restricting elevation from the intended 30° to 10° . To improve this, the bottom of the shin and the top of the foot were both pulled in to decrease interference and increase the foot's range of motion. The change in upwards motion is shown in Figure 4.2.3a and 4.2.3b below.

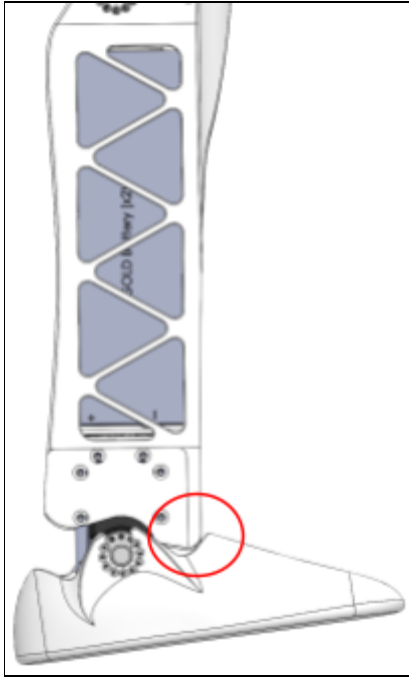


Figure 4.2.3a: Before modifying the range of the foot and shin.

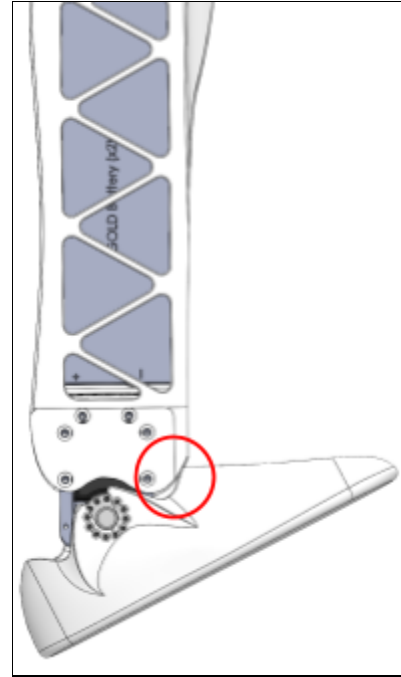


Figure 4.2.3b: After modifying the range of the foot and shin.

With all the modifications, the modified shin is similar to the original, but has increased height, length, modifications to range of motion, and the split design for batteries. The original shin and the modified shin can be seen in Figure 4.2.4 below.

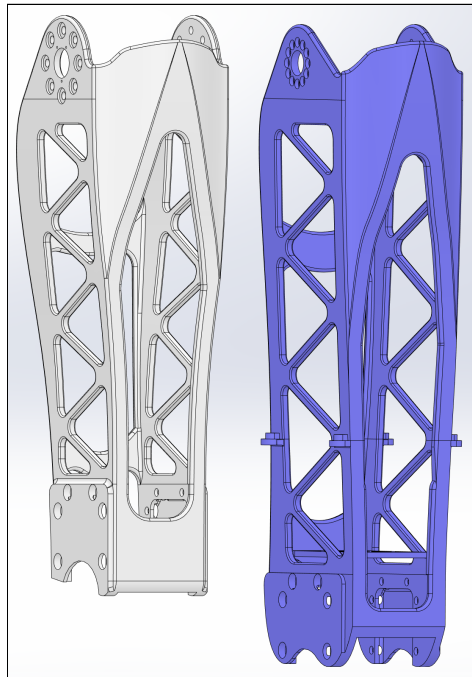


Figure 4.2.4: Comparing the original shin (left) to the modified shin (right).

Modifying the shin's length meant it did not interact with the thigh as it had previously. Shown in Figure 4.2.5 below, the top of the shin is 8.48mm too far forward and 5.29mm lower than the bottom of the thigh.

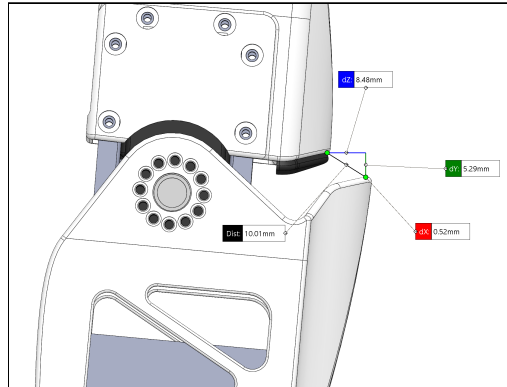


Figure 4.2.5: Gap between modified shin and original thigh

To fix this, the bottom of the thigh was lengthened by 8.48mm, and brought 5.29mm lower. The lengthening is shown in Figure 4.2.6a, and the lowering is shown in Figure 4.2.6b.

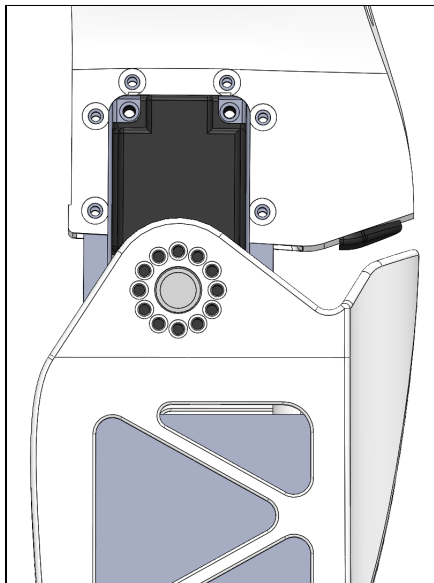


Figure 4.2.6a: Lengthening the thigh so that it matches the shin.

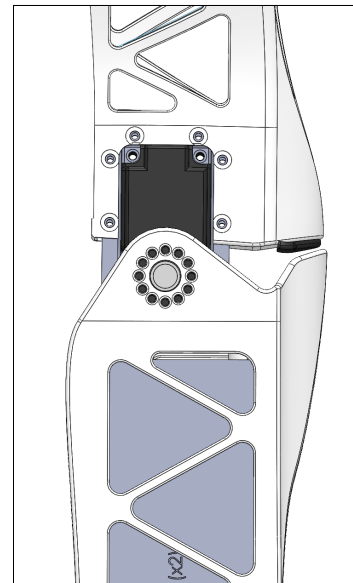


Figure 4.2.6b: Lowering the thigh so that it matches the shin.

4.2.2 Grasping Hands

As a part of the objectives, the team aimed to give Koalby grasping hands. Grasping hands would increase the capability and applications for Koalby, as he would then be able to pick up and place objects, more similarly to an industrial robot. While not a part of the official Poppy

project, a community member created actuating hands and a wrist for Poppy (Figure 4.2.7). The hand with wrist is made up of 5 Dynamixel XL320 motors.



Figure 4.2.7: Actuating hands on a wrist design for Poppy (reproduced from Sosa (2015))

Since the team only wanted an actuating hand and did not require a wrist, only the hand portion was reused from the arm above. This creates a 1 DOF hand. Since the hand on the arm above connects to a motor a part of the wrist, the team created a forearm connector so the hand can be attached to Koalby's existing forearm. The hand with forearm connection piece is shown in Figure 4.2.8 below.

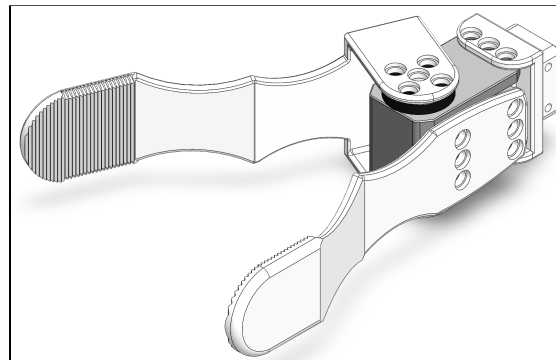


Figure 4.2.8: Actuated hand for Koalby

The team printed and assembled two actuating hands, shown in Figure 4.2.9 below. They are put together using the OLLO Rivet Set RS-10 that is designed to interface with the motor horn (<https://www.generationrobots.com/en/401870-ollo-rivet-set-rs-10.html>).

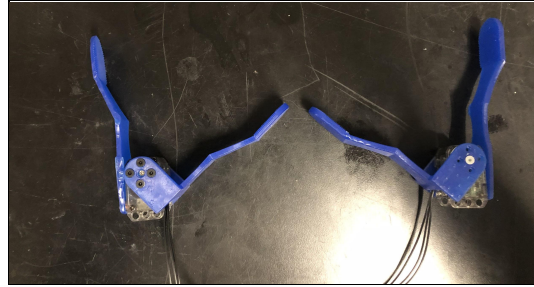


Figure 4.2.9: The assembled actuating hands for Koalby

Due to time constraints, the assembled hands were not attached to Koalby or tested, but they are able to be used by future projects.

4.3 Electronic Design Changes

In order to have full control of Koalby and to make him more functional, the team needed to modify the electronics. The team added three batteries and an Arduino Mega 2560 as a second control board.

4.3.1 Batteries

The original Poppy Project used two 12V power supplies that connected to a wall outlet. The key design change the team added was two 7.4V 5200mAh batteries (Figure 4.3.1) in the shins, and one 11.1V 2200mAh battery (Figure 4.3.2) in the head. All three of these batteries use Lithium Polymer cells. The reason the team chose Lithium Polymer batteries is because they have a high discharge rate. The 7.4V batteries have a discharge rate of 260 A and the 11.1V battery has a discharge rate of 66 A.



Figure 4.3.1: 7.4V 5200mAh Battery (reproduced as is from https://www.amazon.com/dp/B092CZGW2P/ref=cm_sw_r_cp_api_i_VB839KW2VMXDSXXDGA2C)



Figure 4.3.2: 11.1V 2200mAh Battery (reproduced as is from https://www.amazon.com/dp/B08KD1YN9F?ref_=cm_sw_r_cp_ud_dp_4DS7S8H5P9X01JRT50C4)

4.3.2 Control Boards

The original Poppy Project used a Raspberry Pi and a custom built shield (<https://www.generationrobots.com/en/402420-carte-pixl.html>) that can be seen in Figure 4.3.3 below. The team was not able to purchase the shield because it had to be shipped internationally. Since the team could not gain access to this shield, the team decided to use an Arduino Mega 2560 (Figure 4.3.4) with the Dynamixel Shield (Figure 4.3.5). The Arduino Mega 2560 allowed for both the control of the Herkulex motors and the Dynamixel motors.

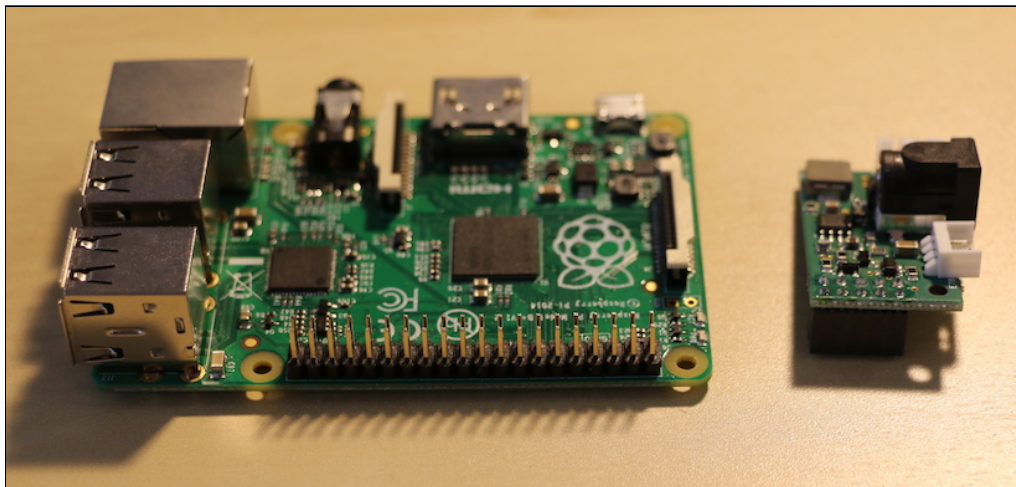


Figure 4.3.3: Raspberry Pi (Left), Poppy Shield (Right)

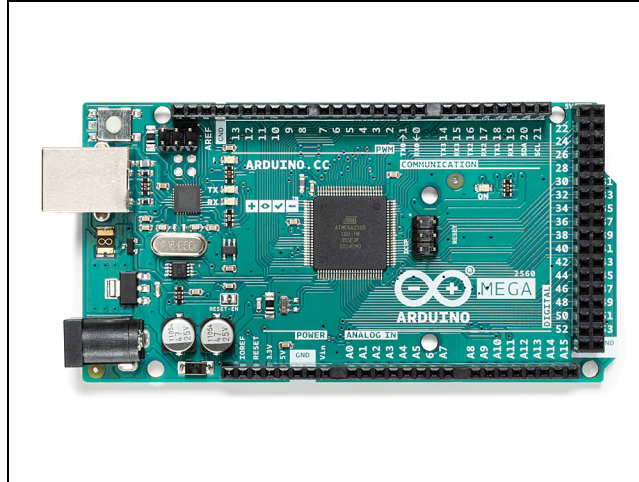


Figure 4.3.4: Arduino Mega 2560 (reproduced as is from <http://store.arduino.cc/products/arduino-mega-2560-rev3>)



Figure 4.3.5: Arduino Dynamixel Shield (reproduced as is from <https://www.generationrobots.com/en/403360-dynamixel-shield-for-arduino.html>)

5 Manufacturing and Assembly

This section looks to expand upon the processes used to physically construct all the components of the robot and assemble the system together once these are gathered. Manufacturing of the parts primarily used resin 3D printing, however a few parts were required to be laser cut out of aluminum. Assembly of the robot itself was done using various fasteners.

5.1 3D Printing

All structural components used on Koalby were resin DLP printed using the Elegoo Saturn model of printer. A full list of all printed parts used in the Koalby robot can be found in appendix F.

Originally, the team used eSun's hard-tough resin product to ensure that the parts would not be brittle and could survive wear and tear, stress from the weight of the motors, and any accidental drops that may occur. After testing with the right arm assembly, it became apparent that the hard-tough resin was too flexible for some of the thinner parts of the robot, specifically the shoulder. While the parts would not break, they would bend and flex such that the motors were put under significant load.

To solve this issue, the team took advantage of one of the benefits of resin printing: ratio mixing. Multiple resins can be mixed in different ratios to give properties between either of the two resins. The team used both the hard-tough resin and a hi-temp resin from eSun, with the latter being much more rigid (specifications for both shown in Table 5.1.1 below). Mixing the more rigid resin with the tough, but more flexible, resin produces parts that are suitably rigid while also being strong enough to not easily break.

Table 5.1.1: Properties of hard-tough and hi-temp resin

Resin	Shore Hardness	Tensile Strength (MPa)	Flexural Strength (MPa)
Hard-tough	81 <i>(Hard-tough resin blue: Esun 3D printing materials)</i>	50-60 <i>(Know Your Materials: SLA Tough Resin: Fast Radius 2022)</i>	70-80 <i>(Know Your Materials: SLA Tough Resin: Fast Radius 2022)</i>
Hi-temp	82-84 <i>(High temp resin)</i>	70-85 <i>(High temp resin)</i>	95-105 <i>(High temp resin)</i>

All of the redesigned parts of the Koalby robot were first sliced using the Chitubox software provided with the printers, and then printed on print beds laid out for minimized print times. This meant arranging parts such that as many smaller parts could fit on the same bed as a

larger print. For example, a print bed containing one of Koalby's entire arms is shown in figure 5.1.1 below.

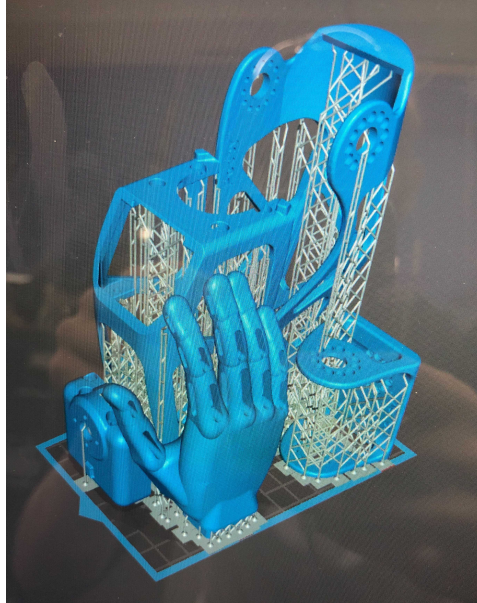


Figure 5.1.1: Right arm print bed layout in Chitubox software

The curing parameters for all prints were the same, except for the head, for which the support density was raised to 60%. The support parameters are shown in Figure 5.1.2 and the curing parameters are shown in Figure 5.1.3 below.

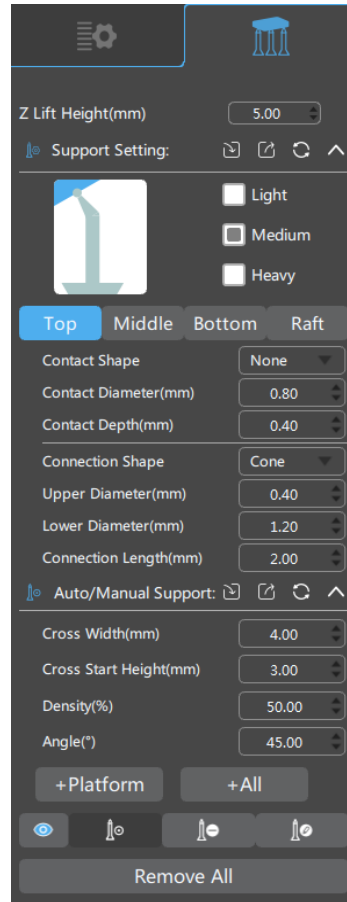


Figure 5.1.2: DLP Chitubox Support Parameters

Exposure Time(s):	3	Bottom Exposure Time(s):	30
Lift Distance(mm)	7	Layer Height(mm)	0.050
Lift Speed(mm/min):	70	Retract Speed(mm/min):	210

Figure 5.1.3: DLP Chitubox Curing Parameters

5.2 Laser Cutting

The MX-64 motors use a separately purchased servo horn (HN05-N102, www.robotis.us/hn05-n102-set/) which costs \$11.90. Originally, the group 3D printed a substitute (Figure 5.2.1), but it was too brittle to work reliably (see section 8.2). This part has a relatively simple geometry, so the group had our own version laser cut to save money and shipping time. Figure 5.2.2 shows the original servo horn.

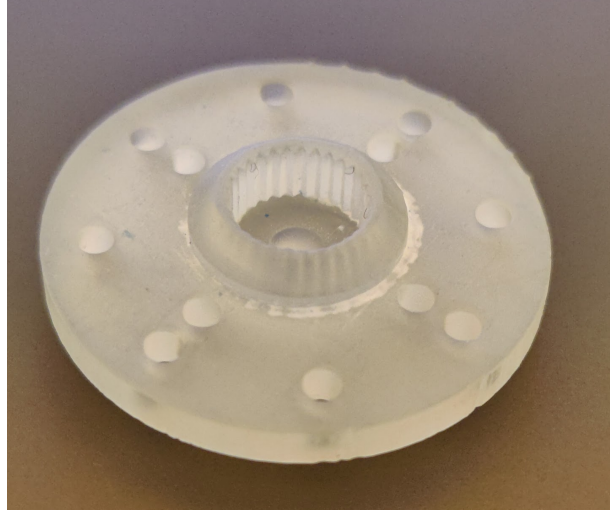


Figure 5.2.1: 3D printed Servo Horn

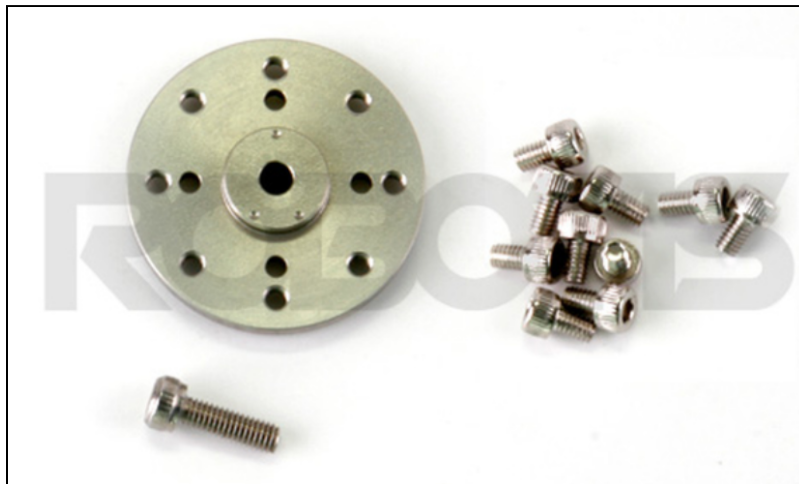


Figure 5.2.2: HN05-N102 MX-64 servo horn

The geometry of this servo horn is not entirely 2-dimensional, as it features a raised washer-like extrusion to house a screw which holds the horn in place. In order to manufacture the servo horn more cheaply, that raised section was omitted, and instead the central screw holds the horn in place with a washer. The 2D geometry is shown below in Figure 5.2.3.

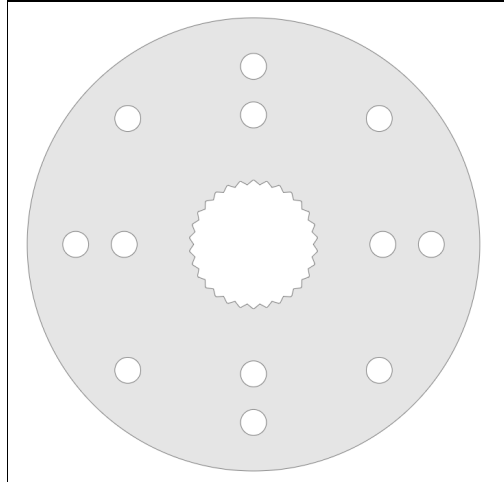


Figure 5.2.3: Koalby aluminum Servo horn

This servo horn was ordered from SendCutSend (sendcutsend.com), which can manufacture 2 dimensional parts from a wide range of materials. The team had the part laser cut from $\frac{1}{8}$ inch thick 7075 aluminum, and ordered 14 for a unit cost of \$2.21, less than 20% the cost of the original servo horn.

5.3 Assembly

Koalby is assembled using 3D printed components, 19 HerkuleX motors, 6 Dynamixel motors of several types, and several hundred M2-M3 fasteners. Full assembly (see instructions in Appendix D) takes approximately 20 hours excluding part print time. The general process requires setting up each motor with the required adaptors and spacers, then attaching the motors to the main body parts. Each main piece is connected to the motors via four M2 screws, where each piece is either at the rotation axis or at the motor mounting points. The total hardware cost of the system is around \$4200, reduced from the \$7000 Poppy Project. Costs are detailed in Appendix B.

6 Electronics

Koalby's electronics differ significantly from the base Poppy Project. With Koalby, power is distributed to motors and the control system from onboard batteries instead of a wall tether, like Poppy. Additionally, the control wiring runs signals to the Arduino from the separate HerkuleX and Dynamixel bus systems. This section will discuss how the electronics of Koalby are configured and how they control the robot.

6.1 Electrical Power

The original Poppy project was powered from a 120V AC wall outlet through a 12V DC converter. This configuration restricts motion by requiring a tether to the wall so the team decided to power Koalby with two 7.4V and one 11.1V lithium polymer batteries instead. The two 7.4V batteries have a capacity of 5200mAh and contain two cells. The third 11.1V battery has a capacity of 2200 mAh with three cells. These batteries can be seen below in Figure 6.1.1.



Figure 6.1.1: The batteries used in Koalby

Since Koalby uses two different motors with different operating voltages, the battery pack needs to supply both 7.4V and 14V. To achieve this, the two 7.4V batteries were put in parallel to power the Herklux motors (which can run on anywhere from 6 to 9 V), Raspberry Pi 4, and Arduino Mega. The Arduino Mega has an integrated voltage regulator, so it can be powered directly from the 7.4V batteries. In contrast, the Raspberry Pi does not have a voltage regulator. Hence, Koalby uses a LM7805CV Linear voltage regulator to power the Raspberry Pi. This regulator supplies up to 1.8A and is sufficient for the Raspberry Pi, which needs 1.6A to operate. In order to achieve 11.1V, the third 11.1V battery is used to power the Dynamixel motors. The wiring configuration can be seen below in Figure 6.1.2.

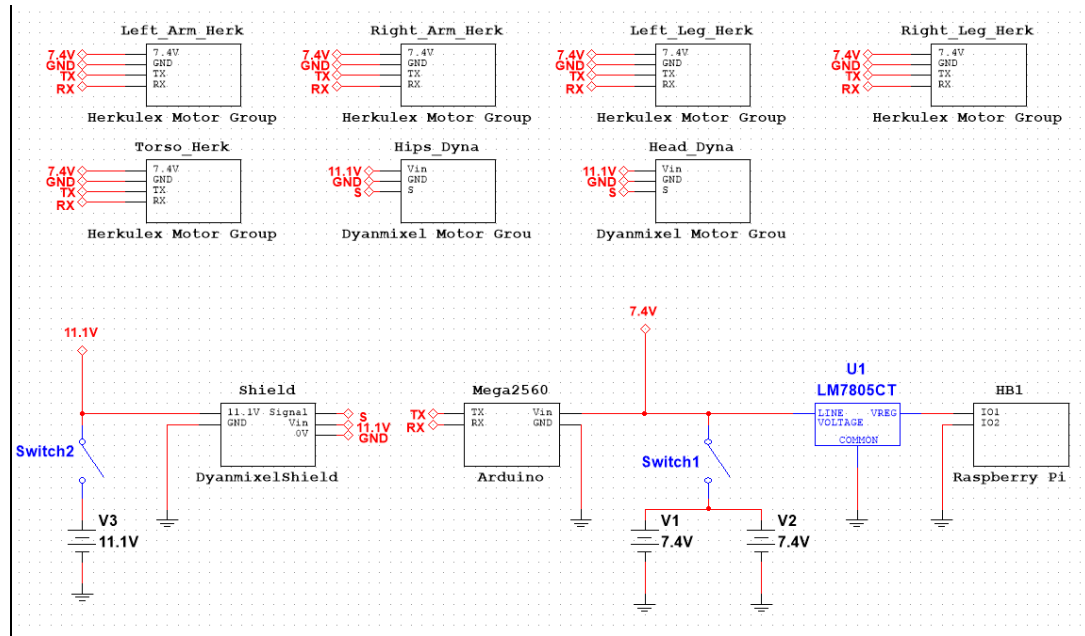


Figure 6.1.2: Power Electronic Schematic

6.2 Electronic Control

Poppy was designed using entirely Dynamixel motors. In the Poppy robot, these motors were connected directly to the Raspberry Pi via a custom built PCB that connected directly to the GPIO pins. This architecture is shown in Figure 6.2.1. This board is not easily available in the United States and the HerkuleX motors use a different four wire bus standard than the three wire Dynamixel setup. The team redesigned the electronics for Koalby to support multiple motor bus systems.

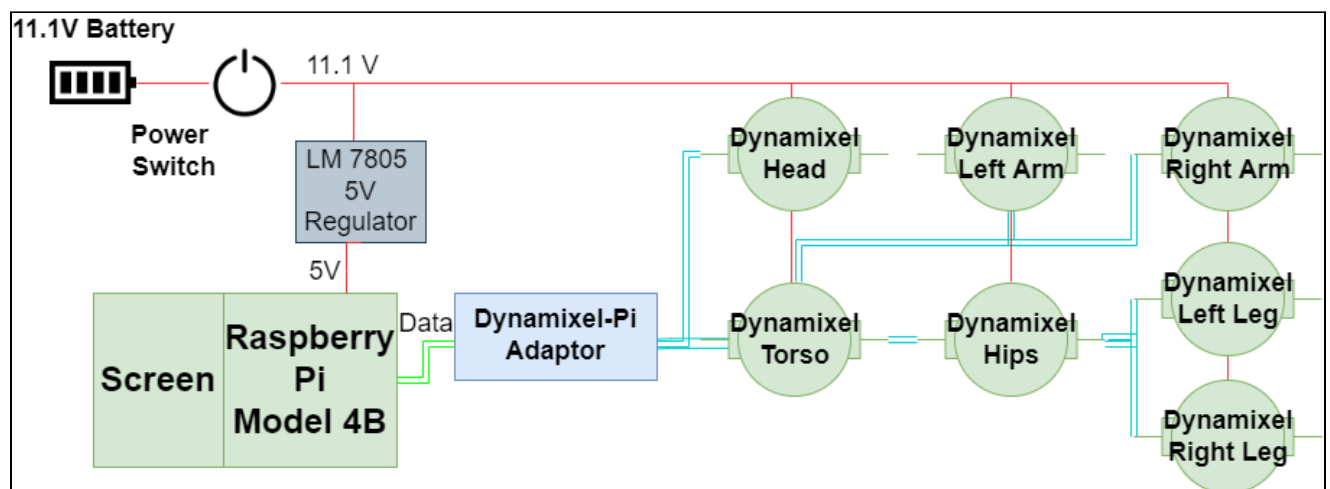


Figure 6.2.1: Poppy Electronics Setup

The team decided to add an Arduino in place of the smaller adaptor board. Using a microcontroller means that messages from the Raspberry Pi can be interpreted and set to the relevant motor bus. HerkuleX motors can be controlled directly from the Arduino, while Dynamixel motors require an additional shield. The Raspberry Pi will be responsible for the same high level control it is in the Poppy architecture, while offloading some of the motor control to the Arduino microcontroller. The full design structure is shown below in Figure 6.2.2.

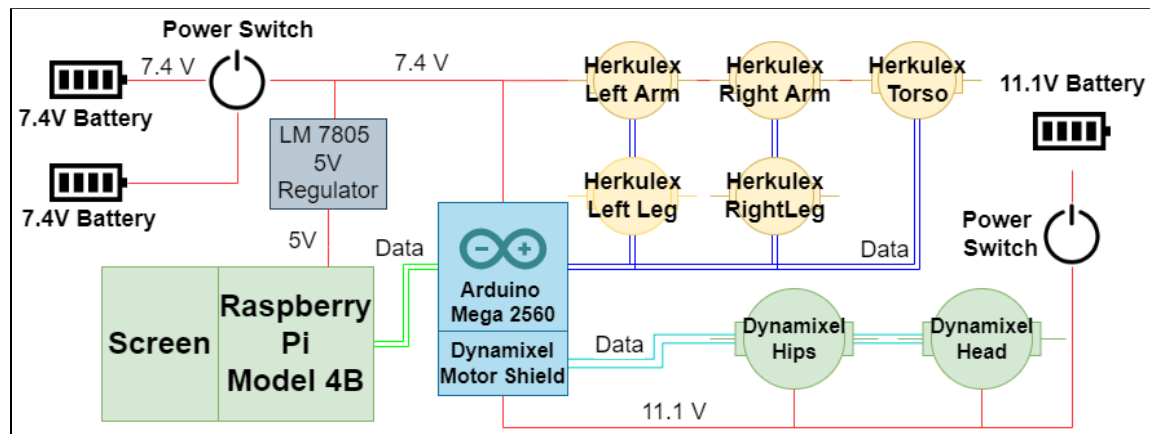


Figure 6.2.2: Koalby Electronics Setup

This new electronics setup will require additional space to store the relatively large Arduino, which cannot fit in the original Poppy head (Figure 6.2.3a) alongside the Raspberry Pi. An alternative design for the head was developed to contain Koalby's larger electronics. The new head design (Figure 6.2.3b) is based around the maximum print size of the Elegoo Mars 3D printer, giving it a boxy shape. This head design provides significant internal volume, and is able to comfortably store the Arduino and Dynamixel shield, the Raspberry Pi, and the small 11.1V Dynamixel battery, as shown in Figure 6.2.4 below.

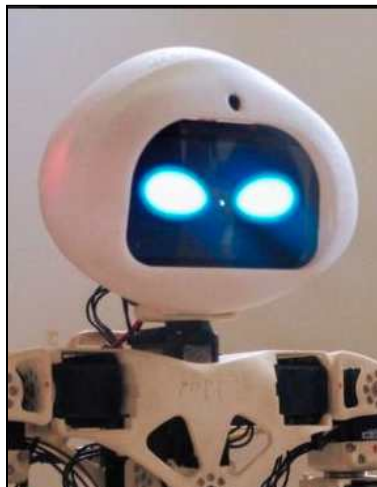


Figure 6.2.3a: Poppy EVE head



Figure 6.2.3b: Koalby's Modified Head

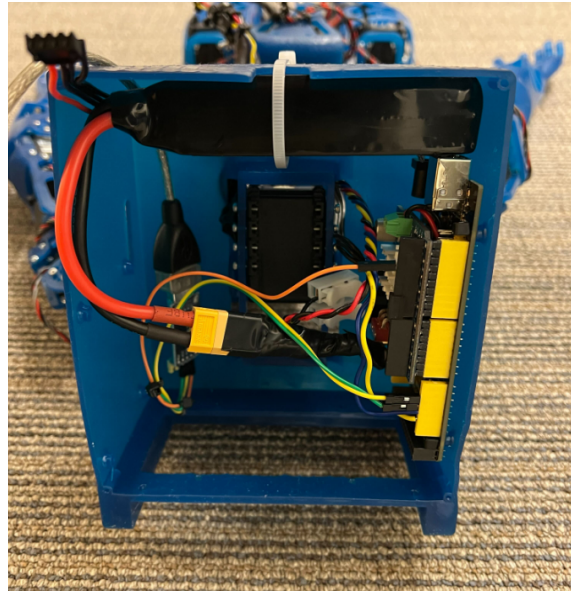


Figure 6.2.4: Internals of Koalby's Modified Head

This requires that the underlying firmware running on the Pi be changed to support communication with the Arduino rather than the Dynamixel motors directly. The Arduino will listen for serial communication from the Raspberry Pi over USB, and interpret those messages to communicate with the correct motors. This abstracts the specific motor type from the perspective of the Raspberry Pi, as the Arduino will select the motor (type and address) to communicate with based on an assigned identifier agreed upon with the Raspberry Pi - this identifier can remain constant even if the physical motor is changed in any way - such as its bus ID being reset or even being replaced by another motor.

Serial communication is accomplished via a USB-serial adaptor. This is necessary because the Arduino uses the *Serial* pins for the USB port. The Dynamixel shield is designed for the Arduino Uno, which features only 1 serial port, and as such is restricted to using *Serial*, disabling the USB port. The adapter connects pins from the Raspberry Pi's USB port to the *Serial2* pins of the Arduino, as the separate serial lines can be used simultaneously to allow for communication with different devices.

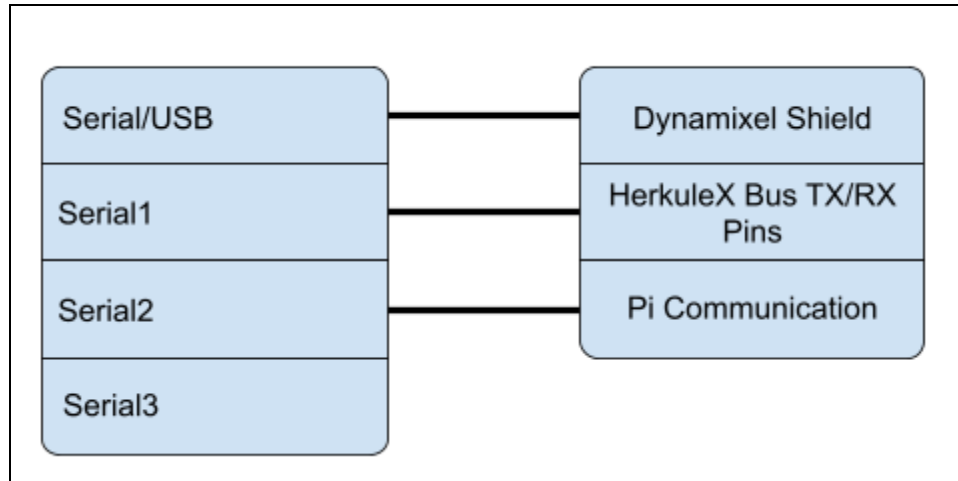


Figure 6.2.5: Koalby Arduino Serial Port Usage

In total, the Arduino uses 3 of its 4 serial ports (Figure 6.2.5). *Serial* is used for the Dynamixel Shield. The shield attaches to the top of the Arduino, and contains circuits to combine the separate TX and RX pins of the serial port into a single signal pin for the 3-wire Dynamixel bus. The HerkuleX motors use the *Serial1* pins. These motors use a 4-wire bus with 2 signal pins, and as such they connect directly to the Arduino TX/RX without needing any adaptor circuitry. *Serial2* is used for communication with the Pi through the adapter, and the Arduino listens on this port for communication during regular operation.

7 Coding

Koalby's code is made up of two components: low level Arduino firmware and high level RaspberryPi control code. The Arduino code, in C++, interfaces directly with the motors, controlling them with signals through its serial ports. The Raspberry Pi code, in Python, communicates with the Arduino and performs high level tasks by sending a sequence of commands.

7.1 Arduino Firmware

The Arduino code is based around a two part continuous loop. In each iteration, it first checks the serial buffer for communication from the Pi, and then performs any maintenance tasks. If no command is available, the state machine loops again rather than continuing to wait. This ensures that any tasks in the maintenance part of the loop are executed regularly, even if the robot is idle.

Commands are sent in a comma separated integer format. The first parameter is the command. These numbers are specified in an enum in the firmware. The available commands are shown in Table 7.1.1 below.

Table 7.1.1: 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

Each of these commands takes a variable number of parameters. Once the command is received, the Arduino waits for the correct number of values to be passed in before continuing. In typical usage, the Pi will send the entire command at once, and delay will be negligible. If this were not the case, this waiting would disrupt maintenance tasks. Because both the Pi and Arduino are being programmed together, this system can work with the Pi responsible for sending data in a single message instead of sending partial commands. This reduces the need for error checking of input data.

The *init* command is sent by the Pi on startup, or when resetting from a failure state. It takes no parameters, as the Arduino motor definitions contain the data required to initialize each motor. This replaces much of Poppy's python motor definition and initialization process.

The *GetPosition* command returns the position of a given motor. It takes one parameter, the motor's communication ID. It returns the motor's position. This is mapped to a 0-100 integer value, with 0 and 100 as the limits of the joint's motion. The range can be variable, so the Pi holds a list of motor limits in terms of angle from a certain reference, and translates that to the 0-100 value. This communication value is useful as motors are not always zeroed to a logical position. By storing the actual motor values on the Arduino and the limits in reasonable real world terms on the Pi, this intermediate value translates easily between the two.

The *SetPosition* command takes two parameters, motor communication ID, and target position. The target position is in the same 0-100 range as *GetPosition*, and the motor moves to that position at a default speed set so that the motion will take one second. The *SetPositionT* command functions just like *SetPosition*, but it has an additional parameter to set the time of motion in milliseconds.

The *ArmMirror* command maps the values read from one arm's motors to the other. The right arm motors are disabled, so that they move freely. The left arm motors are set to positions calculated by reading the right motor positions and mapping between their respective limits. This command will be located on the Pi in the future, however its presence allows for testing key features of the Arduino code, including ensuring the state machine can perform regular updates while dealing with other commands.

The *SetTorque* command sets the torque of a motor to on or off. When on, the internal controller of the motor will actively maintain the desired position. When torque is off, the motor rotates freely.

The *SetCompliant* command allows motors to be manipulated without letting them be moved by gravity the way disabling torque would allow. The Dynamixel's support this feature natively, but the HerkuleX's do not. The Arduino uses a pulse width modulation (PWM) based approach to simulate this. With PWM, the motors are turned on and off intermittently at a high frequency to provide an illusion of fractional torque. For example, a motor on half the time and off the other half behaves as though it is at half of its maximum torque output. Each compliant motor is added to a set of motor IDs. During every iteration of the control loop, the motor has its target position set to the current position. This means that when displaced, the motors will hold at the new position instead of returning to their original position. In order to reduce the torque

required to move the motor, the motor's torque is turned off for some iterations of the loop, and on for others. As the loop executes rapidly, it creates the illusion of reduced but constant motor torque.

The Pi code utilizes these basic commands to translate high level processing into simple discrete steps, which are sent to the Arduino which directs motors to perform those discrete steps while the Pi performs high level processing.

Motors are stored in an array of structs. Each motor struct contains the information that the Arduino needs to translate the commands from the Pi into messages on the motor bus. Table 7.1.2 shows the content of each motor struct.

Table 7.1.2: Koalby Motor Definitions

Parameter	Description
Communication number (implicit from the array)	The number used by the Pi to indicate which motor to access. This is the index in the array of that motor, and does not inherently relate to its hexID. It is unique
HexID	The motor ID, a hexadecimal number. This can be duplicated, where the same ID is used for a HerkuleX and Dynamixel
MinPos	Minimum angle of the motor, corresponds to a 0 command
MaxPos	Maximum angle of the motor, corresponds to a 100 command
HomePos	Home position of the motor, used during initialization
Type	Type of the motor, HerkuleX or Dynamixel. This allows the Arduino to send commands on the correct motor bus

Each motor has a unique communication identifier assigned to it. This corresponds to the index in the motor array, and as such it is sequential starting at 0. These are referenced by the Pi when sending commands directed to specific motors. When the Arduino receives the command, it retrieves the motor's data from the struct in that motor array.

The *HexID* is the address assigned to the motor itself. This is the value used by the Arduino when sending commands to the motor. This is a separate value from the communication ID because of the separate bus systems. The messages to HerkuleX and Dynamixel motors are sent on their respective buses, and because of this, a HerkuleX motor and a Dynamixel motor can share the same hexadecimal ID number without conflict. To avoid this ambiguity of which motor is being referenced by the hexadecimal ID, the Pi refers to motors using a unique communication ID. The Arduino can then use this ID to look up the motor information, which includes both the motor's address and its type, which is sufficient to uniquely identify a motor. This also allows for motors to be replaced by new motors using different addresses without changing the Pi code.

The motor limits are often unintuitive as the HerkuleX motors have no clear way to visually determine them. To account for this, all communication between the Pi and the Arduino treats the motor position as a value from 0-100. *MinPos* and *MaxPos* are the encoder angles of the motor's minimum and maximum range of motion. These limits define the "0" and "100" positions respectively. *HomePos* is also an angle reading, but corresponds to the motor's initialization position. Often this is halfway between the motor limits, but in many cases it is not.

The final value in the struct is the motor type. This can be either HerkuleX or Dynamixel and is used by the Arduino to determine how to communicate with the motor. This could be modified to support an arbitrary number of actuator types by adding values to this field. For instance, if the hand is controlled by small hobby servos, these could be added to the same motor array and treated as any other motor, even though their actual behavior and control mechanisms are significantly different.

7.2 Python Control Code

The Raspberry Pi code can be described by a few subsections. The first was the basic communication between the PI and the Arduino. Setting this up allowed for any high level code or sensors running on the PI to affect the motors connected to the Arduino, and the PI to get readings from the Arduino on motor values. The second subsection of the code was establishing the kinematics of the robot in the code. By doing this, certain functionalities would become simpler to implement, especially any high complexity motions in the future. The third and last subsection the code can be divided into is the primitive system that the team worked to replicate from the Poppy project. This system would enable the scheduling and combining of commands just like in the Poppy project, which would be important for tasks like self-balanced walking if any future projects wish to expand upon what has been done with Koalby thus far.

7.2.1 Pi/Arduino Communication

In order for Koalby's Raspberry Pi 4B to communicate with the Arduino, it must be plugged in with a serial usb cable. This requires the Raspberry Pi 4B to physically be on the robot, which poses a problem for communication between Koalby and a user's computer. The original Poppy used a custom operating system with a Wi-Fi enabled hotspot that controlled the robot. The team decided instead, it was best to set up a Wi-Fi hotspot and use a standard operating system for the Raspberry Pi on Koalby. This would allow Python scripts to run directly on Koalby's Raspberry Pi 4B and would not require a custom human machine interface for controlling the robot. The full setup instructions for the Raspberry Pi can be found in Appendix E.

The first problem the team needed to solve was creating a Wi-Fi hotspot. The easiest way to have a Wi-Fi hotspot would have been to buy an inexpensive router and connect devices to it. Instead, the team used a second Raspberry Pi 3B+ as a hotspot because this route was more cost

effective, only costing the team another \$30 for the Pi. Additionally, using a second Pi as a hotspot was more time effective too because we used an existing Raspberry Pi hotspot OS to quickly set it up. Using the Linux `hostapd` (<https://packages.debian.org/stretch/hostapd>) and `dnsmasq` (<https://launchpad.net/ubuntu/+source/dnsmasq>) packages, the team was able to configure a Raspberry Pi 3B+ to act as a hotspot. For a website interface, the team used the RaspAP hotspot software (<https://raspap.com/>). The website interface can be seen in Figure 7.2.1 below.

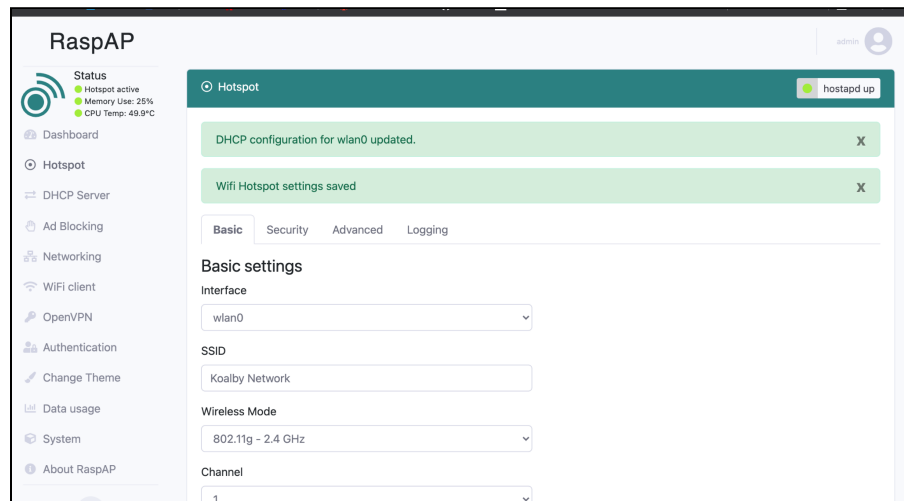


Figure 7.2.1: Raspberry Pi 3B+ RaspAP Website Interface

Once the team had the hotspot setup, the team needed to create a way to remotely control the Raspberry Pi 4B with our computers. To do this, the team used a virtual network computing (VNC) software called VNC Viewer (<https://www.realvnc.com/en/>). VNC Viewer is a cross-platform screen sharing system that was created to remotely control another computer. The VNC Viewer software interface controlling the Raspberry Pi 4B running on a MacOS computer can be seen in Figure 7.2.2 below.

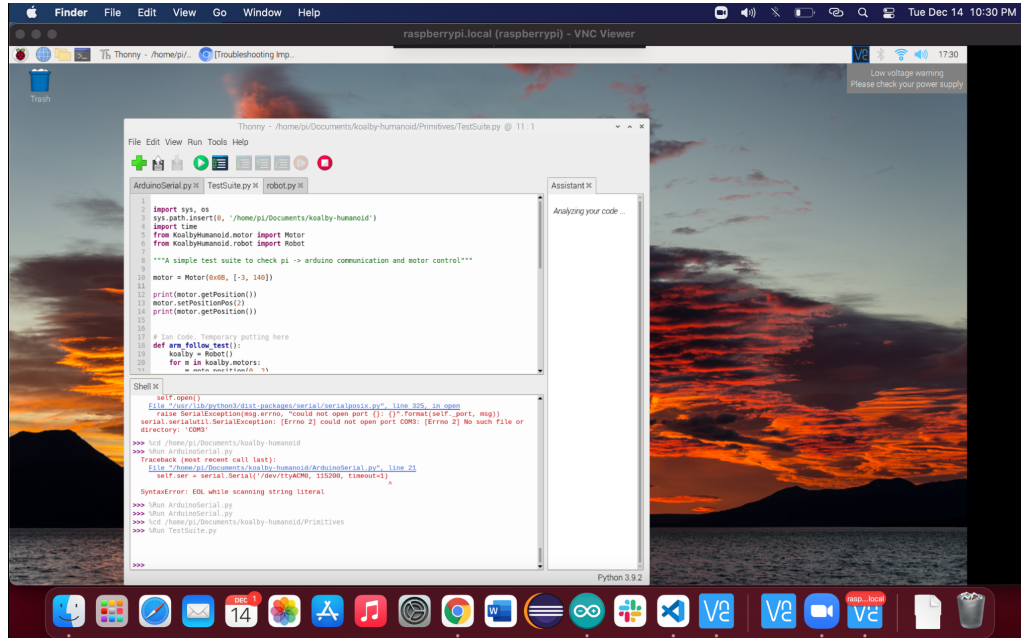


Figure 7.2.2: VNC Viewer running on Raspberry Pi 4B

7.2.2 Robot Kinematics

Precise control of multiple degree of freedom robots depends on knowing the kinematics of each joint chain. The complex math required by kinematics can be simplified by code. The team studied how the Poppy project implemented kinematics and found that they were contained in the library *ikpy*. This library calculates both forward and inverse kinematics automatically, and was built to integrate with the Poppy code. The team was not using the Poppy codebase due to its unnecessary complexity and poor documentation, but the similar structure of the team’s new code should have been able to integrate with this library. Any classes that the library would call for were already being implemented in the structure of the new Koalby code.

This meant that the biggest changes would come in the *URDF* file and the Robot class. The *URDF* file defines the relative dimensions and orientations of each joint. Using a *URDF* file, the inverse kinematics generate all the Denavit–Hartenberg (DH) parameters used for forward and inverse kinematics. This *URDF* file would just need updated joint locations according to any changes in lengths due to the remodeling for HerkuleX motors.

The Robot class is important to the kinematics because it is where the kinematic chains are created. For example, one chain might be the left arm chain of the forearm, upper arm, arm connector, and shoulder as the links, and the motors between these points as the joints. These chains are created through the *URDF* file and functions in the *ikpy* library, but the declarations for the chains are in the robot class for any other classes to reference and use in the future.

The team did, however, run into an issue when working to implement this library. When generating the forward and inverse kinematics on the robot, the code would return an invalid division error within the *scipy.optimize.minimize()* function. Looking into this error, the team

found that code that worked on scipy version 1.4.x may sometimes error on version 1.5.x and this error code was a sign of this being the case. This meant that the code that originally had worked for the Poppy project kinematics would no longer work on the latest versions without modification. Given the time spent on this issue, and the future work, it would require to either rewrite the code to get around this new error, or create inverse kinematics code from scratch, the team decided it was best to shelve this feature for future work in order to accomplish more the pressing goals of the project.

7.2.3 Koalby Primitive System

The Poppy project implements a control system called a primitive manager. Looking further into the code, the team worked out how this primitive manager and primitives system worked. The original Poppy creators had implemented the system to quickly merge commands through a multi-threaded queue. The primitive system works as a kind of interceptor for any command sent to the robot. Whenever a command, like setting a motor to a position, is sent to the robot through a primitive, the primitive manager logs it in a queue instead of sending it directly to the robot. Once in the queue, it will then merge the command with other commands of like-type that are in the queue at the same time, averaging their values for a merged output that gets sent to the robot. This output value can be changed based on the merging filter set, which in the original code is an average of all queued values. In practice, an example may be that one primitive tells the robot to walk forward without any sensor input, moving the joints to predetermined points. Simultaneously, another primitive reads in pressure sensor data from the feet and tells the robot to move the left leg to a certain position to maintain balance. These two sets of motor positions would then be caught by the primitive manager and merged such that the actual position that the robot moves its left leg to is an average between the two positions, allowing it to rebalance, but still move forward. The team originally speculated that this system was implemented by the creators to help with position corrections while walking, however, the primitive manager could be useful for any merging behaviors.

The team began editing the original primitive system to work for the new Koalby library. First, much of the original code could be trimmed down to remove anything pertaining to direct dynamixel control, then edits were begun to update methods such as *gotoposition()* and *setgoalspeed()*. After the basic edits were made, the team began to work to change the *MockupMotor* and *MockupRobot* classes that are used to contain the fake motor data used to allow queuing.

At this point, the team realized that it may be easier to trim down the primitive manager. This new version would be a class that would similarly run a queue of values to be merged and would have the ability to be toggled on or off. However, this new class would not need to implement the *MockupMotor* or *MockupRobot* classes as it could set values to motors without actually sending them to the robot through the Arduino. This would simplify the system to a timed queuing dictionary that updates the physical robot over a predetermined time interval with

the merged values from anything queued during that time. This could be separated into motors and sensors as two separate dictionaries to allow for future changes.

In implementation, Koalby's primitive manager was made to merge the dictionaries of multiple different motion primitives to output a final filtered value, which was an average of all input values in this case, that would then be sent to the physical robot. Each individual primitive class inherits the basic primitive fields from a *KoalbyPrimitive* class. These fields are the motor dictionary of all relevant motors and their positions for that primitive motion, all sensor indexes and their values, and an *isActive* boolean value that can be used to enable or disable individual primitives from the robot class.

When in use, Koalby runs multiple threads, including one thread per primitive running, a thread for the primitive manager update cycle, and a master thread. Koalby's primitive manager update method then references all active primitives, reads their motor dictionaries, and then merges those values and sends them to the physical robot. This can be seen in Figure 7.2.3 below.

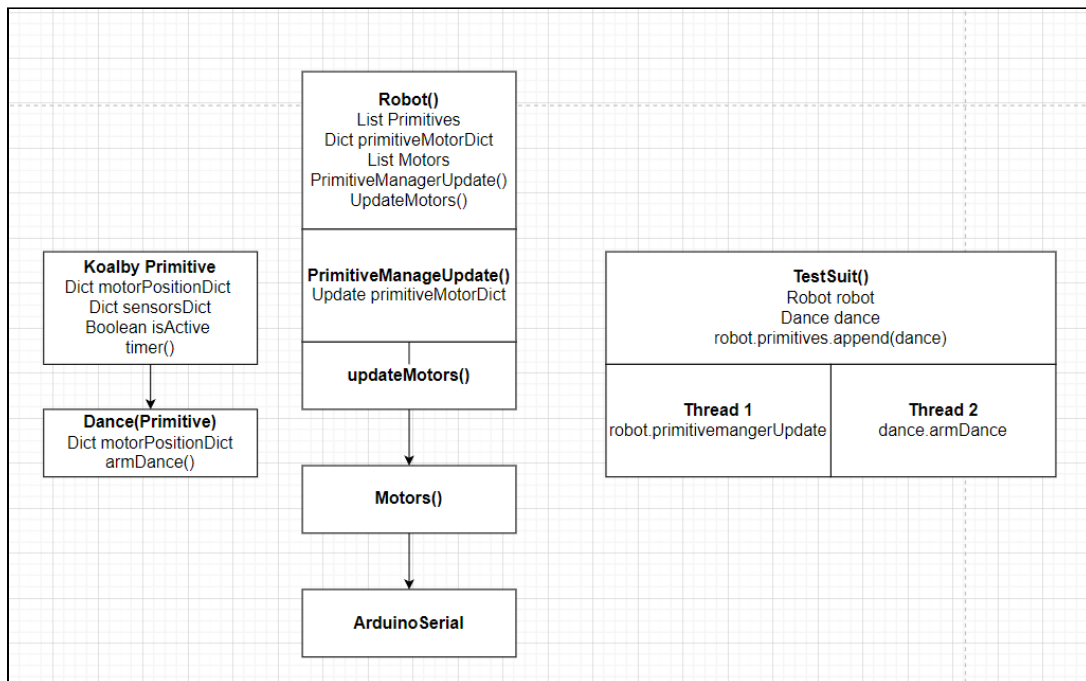


Figure 7.2.3: Flowchart of the Primitive Manager

7.2.4 Record / Replay Primitive

The most useful primitive the team developed was the record/replay primitive. This primitive consisted of two main parts: the *recordMotion* method and the *playMotion* method.

The *recordMotion* method is a utility method used for recording animated pose-based motions and is not run through the primitive manager. To achieve this, *recordMotion* first requests a user input of the number of poses to be recorded in a motion set. It then sets all motors on the robot to compliance so that the user may move joints freely. From this point on, the method runs a loop that awaits user input on each run through. When the user has posed the

robot in the desired position, they type any integer into the console and hit enter. The method will then read all motors on the robot at this position and log them into a motors dictionary that is added to a list of poses. This process is then repeated for all subsequent poses until the entered pose number has been met. At this point, the method will then ask for a desired file name input, parse the list of motor dictionaries into a CSV file format, and save this CSV file for later playback. Poses from of these recorded motions, the wave and handsOnKnees motions, are shown in figures 7.2.4a and 7.2.4b below.

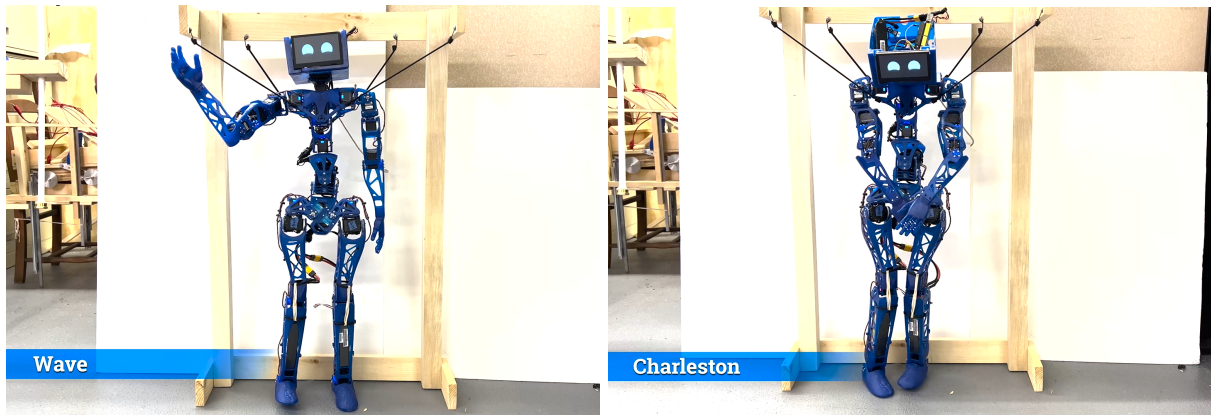


Figure 7.2.4a: Recorded wave Motion

Figure 7.2.4b: Recorded handsOnKnees Motion

The *playMotion* method is a primitive method which is meant to be run using the primitive manager system. The method will first request three inputs: a file name, a pose time, and a pose delay. Pose time refers to the time of which the robot will spend moving from one pose to the next (i.e. speed of motion), whereas the pose delay refers to the time the robot will wait in between motions. Once these inputs have been received, *playMotion* will then parse the CSV file requested into a list of motor dictionaries, and then send each motor dictionary to the primitive manager at the desired times with the desired speeds. There are currently two main use files for this method: *ReplayPlay* and *ReplayPlayLoop*. When run, *ReplayPlay* will request the filename, pose time, and pose delay, then play the requested motion a single time. It will then request the inputs again and wait until they have been given to play the next motion. This is for single action motions, such as the dab motion. Alternatively, *ReplayPlayLoop* can be used to play motions with a number of iterations. When run, filename, pose time, and pose delay inputs will be requested as before, but now a number of iterations will also be requested. The number of iterations will determine how many times the robot will play through a motion with the same parameters before asking for a new set of inputs. Videos of the motions developed using this method can be found in Appendix H.

7.2.5 Multi-Threading and Test Files

The original Poppy project had a library called *pypot* that had three files for multithreading. These multithreading files were very complicated as Poppy had the ability to

start and stop different threads. Initially, the team decided we were not going to use multithreading as it added complexity to the project, but later decided we needed to use it.

In Python, multithreading works by creating a function that loops forever and then having a thread point to that function to run it forever. The issue with this is that once a thread is stopped, the program cannot restart a thread. The way used to get around this issue was to have each primitive either active or not. This allowed a thread to keep looping, but have the code inside be skipped if not needed. Initially this method worked. Since the primitive manager needed to update as fast as possible it was on its own thread. Consequently, every other primitive needed to be in its own thread. The problem with this solution is that once more than 3-4 threads were running, the code started to visibly slow down because the hardware could not keep up with the tasks running. For demonstration, the test files created would have one thread for the primitive manager and then one thread for the primitive.

In order to fix the problem with multiple threads, a stoppable thread class needs to be created to create a new thread when a primitive is activated and then stop it when the primitive is turned off. This was done in the pypot library, and should be mimicked for future work.

7.2.6 User Interface

Since the original Poppy project had a user interface that was built into the operating system on the Raspberry Pi, the team decided it would be useful to create a basic user interface in Python. The team wanted the user interface to start and stop primitives as well as initialize and shutdown the robot. The team used the Tkinter Python package (<https://docs.python.org/3/library/tkinter.html>) to design the user interface. The user interface had one main window with multiple buttons on it. The user interface can be seen in Figure 7.2.5.



Figure 7.2.5: User Interface Design

Each of the buttons on the user interface would allow a function to start when it was pressed. When the main test file started, it would trigger each of the threads to start. Then, the user interface thread would wait for input and trigger the boolean of another primitive's thread to

start. Finally, the primitive manager thread would take the calculated motor dictionaries and send it to the robot. This allowed each button to control a boolean in a thread to start and stop primitives. The control logic can be seen in Figure 7.2.6.

The control logic shows the main script with the “Main” block. This then starts the “User Interface Thread”, “Dance Thread”, and “Replay Thread”. The “User Interface Thread” has the buttons and text for the user interface. The “Dance Thread” is running the dance method from `dance.py`. The “Replay Thread” is running the replay functions from `replay.py`. Both the “Dance Thread” and “Replay Thread” communicate to the “Primitive Manager” what motors need to move with the robot dictionary.

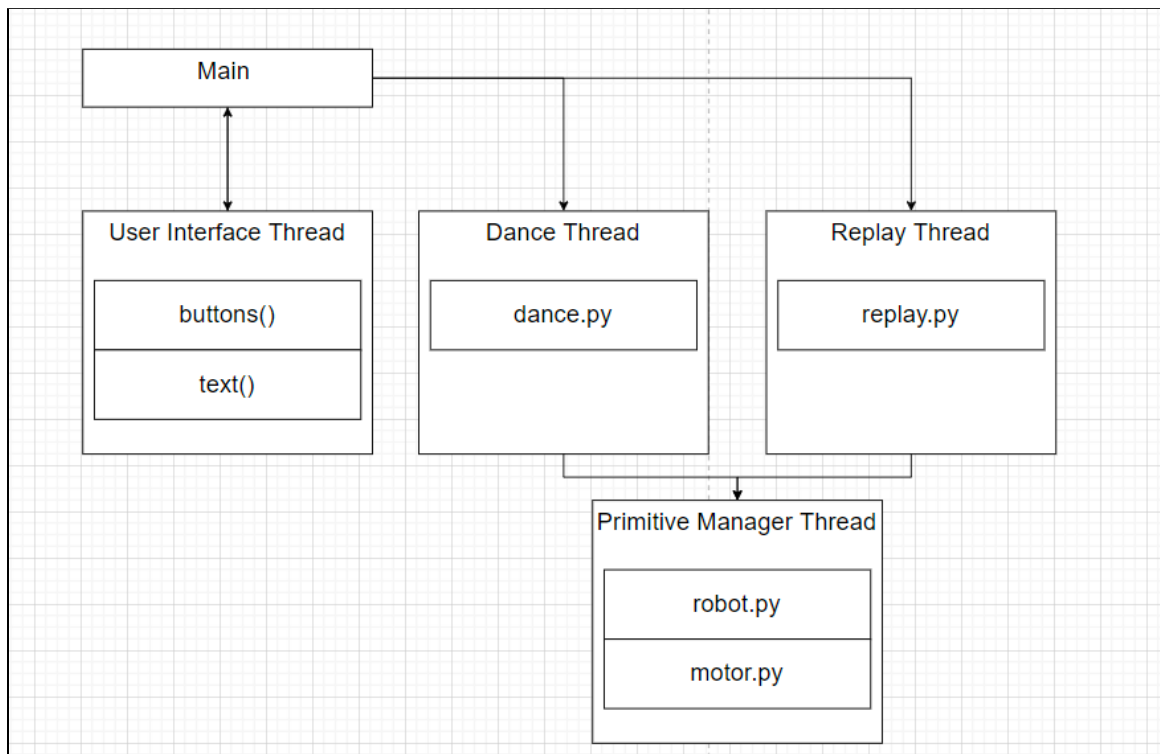


Figure 7.2.6: User Interface Flowchart

8 Testing

Testing for Koalby was conducted through planned testing sessions, code implementation, and full demos of the system. Testing in these methods both exposed Koalby to calculated loads as well as typical usage which helped diagnose and fix any issues with the system and solidify the constraints of the system.

8.1 Arm Testing

The team first tested the right arm. This was the first sub-system constructed, and the goals of testing were to figure out the capabilities of the motors and the code controlling the system. The first tests included a motion test and a load test.

The motion test was primarily a test of the code to ensure motors could be controlled to move to desired positions consistently, and to ensure the range of motion of the system was as expected. To accomplish this test, the team designed a simple wave motion where the robot would raise its arm from its side and perform a wave three times before restarting the cycle. Setting the positions all the motors should move to took a significant amount of time because each motor had a unique range of motion and values it could move to. This meant each position was a trial and error process to code in. Performing this test helped the team determine that it would be important to simplify the code by constraining all motor movements to a mapped range of values as was further discussed in section 7.1. By mapping the motion in this way, the high level code does not need to know the specific encoder values that each motor is rotating to, but only needs to know where in a motors range it should move to. This is something that would have simplified motion testing. Additionally, motion testing also showed the team that a method of recording positions and all the encoder values that make up a position would be extremely valuable to rapidly developing motions. This feature was later added and is discussed in section 7.2.4.

Load testing was conducted in a far more empirical manner. The team attached a small bottle to a piece of rope and attached that rope through the weight saving holes in Koalby's middle finger. The bottle was filled with water to specific weights, which varied and were recorded for each test. Koalby was then to lift his arm sideways and bring his arm upward, moving only his shoulder for a total rotation of 130° . Once the maximum point was reached, Koalby would lower his arm and repeat the test. This setup can be seen in Figure 8.1.1 below. Through this testing, it was found that Koalby could easily lift up to $\sim 175\text{g}$ without issue. However, when reaching 200g , the arm would begin to stall at around 100° of motion. When the HerkuleX motors stalled, they disconnected immediately, so the angle was recorded visually. This testing also revealed that the HerkuleX motors are prone to stalling more frequently after constant use under high load conditions. Overall though, this testing was a positive sign for the robot and the goals of the project. Specifically, the goal of adding graspable hands was directly impacted by this testing. This testing proved that the robot could lift objects if it only had a

method of grabbing them. However that method would have to be lightweight and likely simple to maximize the weight of objects that Koalby could pick up. This is what led to the development of a simple 1-dof grasping hand as explained in section 4.2.2.

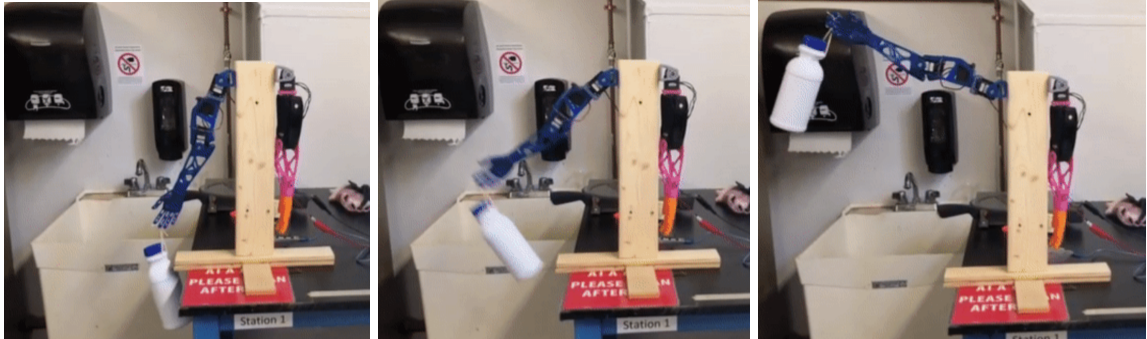


Figure 8.1.1: Weight Testing of Right Arm at ~150 grams

8.2 Routine Testing

While the first tests were conducted as a structured testing session, most of the remaining testing during the project was conducted naturally while developing code and testing it on the robot. The code development process is detailed more in section 3.5 and section 7; however, this testing also helped test mechanical aspects of the system.

One of the first issues revealed through routine use of the robot was failure of the printed parts at hardware mounting points, as shown in Figure 8.2.1 below. Often when only two screws were securing a printed part to a motor, the part would snap due to the unevenly distributed loads at the mounting points. While printed parts are fairly simple and inexpensive to replace, the time cost would delay testing and would come at unexpected times. If this happened right before a demonstration of the robot, it could cripple the functionality. This issue was resolved by ensuring four evenly distributed screws held all printed parts together instead of the two that originally were put in place. Testing with four screws proved the life cycle of these parts was more than sufficient as none have broken since implementing this change. This has included lasting through the bulk of motion development and all robot demonstrations.

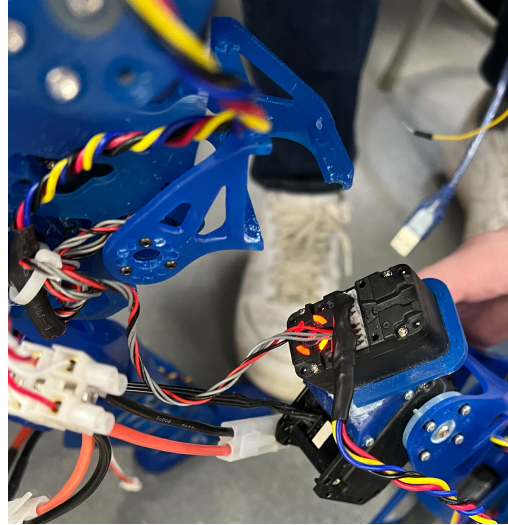


Figure 8.2.1: Broken hip print due to unusual torquing of the leg

Another part that proved insufficient to handle typical robot loads was a printed version of the Dynamixel servo horn detailed in section 5.2. The team wanted to minimize robot cost, and printing these horns was the least expensive option. However, these horns quickly proved insufficient. When working to initialize Dynamixel MX-64 control, bugs in the code caused the motors to jerk to their physical limits. Every time this happened, the printed servo horns would snap. While this action is technically an action the robot should not perform under normal circumstances, bugs in development will happen. It would be acceptable if these parts broke occasionally when experiencing atypical motion, but for them to break every single time indicated that they were not suited for a developing system. This is what led to the laser cut option explained in section 5.2.

Routine testing also revealed the importance of wire management. Wires that were routed through joints out of convenience or lack of attention proved to be problematic when running repetitive motions of those joints. The wires would rub and grind against the printed parts, which would damage the wire insulation. Without insulation, the wires were prone to shorting, causing damage to the electrical systems of the robot. These shorts would cause large delays and costs while electronics had to be replaced throughout the robot. By carefully routing wires away from rotating joints and building in sufficient slack, as shown in figures Figure 8.2.2a and Figure 8.2.2b, the robot is able to operate without risk of damaging any electronics.

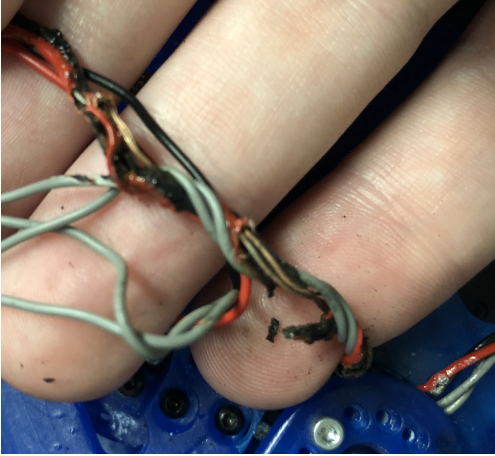


Figure 8.2.2a: Before Image of Faulty Wiring

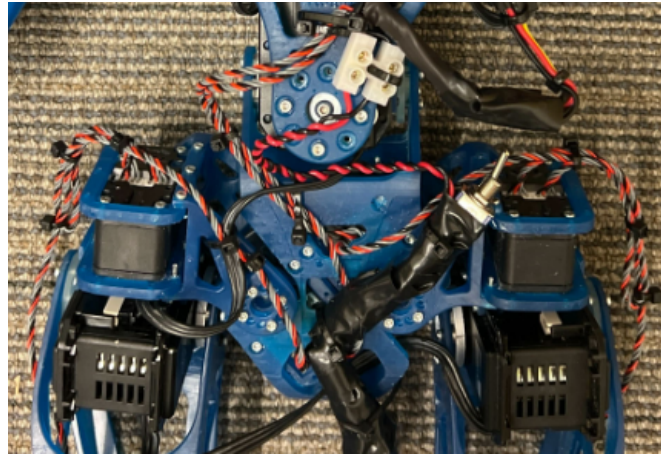


Figure 8.2.2b: After Image of Full Rewire

8.3 Battery Endurance Testing (TouchTomorrow)

While routine testing allowed the physical endurance of parts to be tested, the team needed an extended demonstration to test the endurance of the batteries. The TouchTomorrow event hosted by Worcester Polytechnic Institute (WPI) gave the perfect opportunity for such testing. This event ran for roughly 7 hours and the robot would need to be running continuously as a constant stream of people wandered by and approached the team to learn more about the project. Originally the Poppy project had operated with the use of a powered tether which would of course have no issues with an event such as this. However, one of the major goals of the project had been to eliminate that tether through the use of onboard batteries. The untethered approach primarily extends the walking applications of Koalby in the future by avoiding the need to operate only in precise locations. However, for these batteries to truly be a worthwhile improvement, they also would need to provide power over an extended period of time like the TouchTomorrow demo. The team set a goal of one set of batteries lasting at least half of the event. This would mean that for the robot to operate through the entire event, it would only need to swap batteries once. For longer events one set of batteries could always be charging while another was in use for theoretically unlimited use. Previous routine testing had lasted only a maximum of around two hours, so for this event to be a success the batteries would need to last nearly double their previously demonstrated capability.

The batteries proved a success. Koalby ran continuously throughout the event doing a combination of dancing, waving, marching, and shaking hands until the first set of batteries began to die at around the four hour mark. With this testing complete, the team was able to confidently compare between a tethered and untethered system in regards to their operating capabilities.

9 Discussion

In this chapter, the original project goals will be reviewed, followed by discussions on mechanical design, construction of the robot, Koalby's battery power capabilities, the overall cost reduction achieved by the team, Koalby's abilities with respect to walking, and Koalby's abilities with respect to grasping objects. In addition, all demonstrations by the team and Koalby will be reviewed, highlighting core learning moments gained from each.

9.1 Goals

The goals of this project were to:

- Construct a full humanoid robot based on the Poppy Project
- Reduce total cost by selecting less expensive components
- Power Koalby from onboard batteries
- Give Koalby the ability to walk on its own
- Give Koalby the ability to grasp objects

9.2 Mechanical Design and Construction of the Robot

The goal of constructing a full humanoid robot based on the Poppy Project was achieved. Parts were taken from the open source Poppy project and modified to fit our needs. A majority of modifications came from the implementation of different motors, requiring a motor bracket to be designed and implemented into the assembly. Modifications included lengthening and widening the shins to fit the batteries, and splitting the shin so that batteries could fit inside it, discussed in section 4.2.1. Some difficulties were faced when modifying the existing parts as not all modifications worked well with the parts' designs. One example is the thigh. Due to constraints within the printer's bed size (Elegoo Mars), the thigh was too tall to be printed on it. The team tried modifying the thigh so it could be printed in two pieces. Since this part has a bend feature in its design, modifications to it would either be bent and warped, or would not be allowed by Solidworks due to the part geometry. Since splitting the thigh into two pieces would not be possible, the thigh needed to be printed on a taller printer (Elegoo Saturn).

Robot assembly was successful. However, testing revealed many areas of the assembly process that could be improved. The first version of Koalby was assembled as quickly as possible to enable testing. Due to a shortage of fasteners, the team secured parts initially with only 2 screws instead of the 4 detailed in section 5.3 and Appendix D to avoid delaying testing while waiting for parts to ship. With only two screws, stress concentrated at specific points, causing parts to fail and snap. As a result, the team later dedicated a week to a complete disassembly and reassembly to install the correct number of fasteners. With this new standard,

the parts are held up through all remaining testing and demos without issue, and the four screw requirement is defined in all documentation.

Another challenge with the assembly is with wiring the motors. The Poppy Project parts were designed for Dynamixel motors with ports located on their sides. HerkuleX motors instead have ports on the rear of the motor case, which were sometimes difficult to install or resulted in wires running dangerously close to joints. To resolve this, some wires had to be connected to the motors before the motors were installed in the printed frame as they were inaccessible after installation.

It also was important to route these wires away from rotating joints. During testing, a wire had been mistakenly routed through the rotating section of a hip motor. This wore away the insulation until it caused a major short which destroyed much of the wiring in the lower half of the robot. All of these issues can be avoided by spending extra effort in initial assembly to ensure the wires never run through the rotation portions of the robot.

9.3 Battery Power

The original Poppy design was powered by a wall mounted tethered power supply. This power delivery option limited Poppy's ability to walk far away from walls and also hindered its ability to walk on its own by posing an entanglement risk. Since Poppy was constrained by the wall mounted power supply, the team decided early in the project one of the goals for Koalby was to make the robot fully battery powered. Doing so would allow Koalby to have more flexibility when moving around because the robot would not have to be tethered to a wall for power.

Koalby uses three batteries to power all the motors and electronics. Koalby uses two 7.4V 5200mAh Lithium Polymer batteries with a maximum discharge rate of 250A. These batteries are located in the shins, and power the Herkulex motors, Arduino, and Raspberry Pi. The location of these batteries can be seen in Figure 9.3.1. Koalby also uses one 11.1V 2200mAh Lithium Polymer battery with a maximum discharge rate of 100A. This battery is located in the head and is used to power the Dynamixel Motors.



Figure 9.3.1: Location of 7.4V Batteries

9.4 Cost Reduction

The largest source of cost reduction was using HerkuleX 0201 motors rather than the Dynamixel MX-32 motors used in the Poppy project. This reduced the cost by ~\$2,500 by itself (Appendix B). Despite their significantly lower price, the HerkuleX motors worked comparably to the MX-28's they replaced.

The HerkuleX motors' torque was sufficient to replace the MX-28 and the team designed an adaptor mount and spacer to easily insert the new motors into the design. HerkuleX motors were easy to implement electrically as their wiring busses can be daisy-chained, and the signal wires can directly connect to an Arduino serial port. Some complexity was added to the system due to needing two different operating voltages for the HerkuleX motors and Dynamixels, however this complexity was worth the cost savings.

The biggest challenge in installing these low cost motors was developing code for them. If Koalby used the original MX-28 motors, then the Poppy project codebase could have been usable, significantly reducing initial setup time. However, expanding off of this code could have proved to be difficult based on the scale and complexity of the codebase. Issues like the version conflict found in inverse kinematics (section 7.2.2) would have still been present, leaving the team to debug the already written code. This makes it difficult to analyze the programming benefits of choosing to stay with Dynamixel motors. The team's newly designed programming framework has some advantages over the legacy code. Its scope is smaller, aiming to control Koalby alone instead of providing a flexible framework for multiple robots with computer simulation capabilities. This narrow initial scope makes the team's new framework more readable and easier to debug. By designing around the addition of HerkuleX motors, the system should allow for any motor from standard servos to new types of smart motor to be substituted in with far more ease than with the Poppy project, allowing for continuing improvement as motor

technology continues to develop. Despite the fact that the old and new programming systems cannot be fully compared, the benefits of the new system again point towards the motor substitution being a worthwhile and successful change.

Using resin printing versus other forms of manufacturing also played into the final cost of the full assembly. While it is difficult to compare this cost to other models and versions of the Poppy platform, it can be said that due to the reliability of resin printing, cost of failed prints was minimized. The total cost breakdown of the printed parts of the robot can be found in Appendix G. 3D printing allows precise geometries to be manufactured at low cost, and as per Appendix B, resin costs make up less than 2% of the robot's cost.

9.5 Walking

While automated walking and self-balancing were originally goals of this project, these proved to be outside the scope of the available time frame. Instead, the team opted to aim towards the beginnings of assisted walking cycles using the replay primitive.

Through this method, two early versions of assisted walking were achieved. Both versions of walking require at least half of Koalby's weight to be supported externally, either through use of a stand and harness, or through user help. The first walking cycle (`raiseKnees.csv`) allows Koalby to take very small steps backwards, using only the knee and ankle motors in the legs. The second walking cycle (`wk3.csv`) implements all leg motors and allows Koalby to take a number of steps forward equal to the imputed iteration number multiplied by two. One issue that arose during these walking tests was the ankle strength. For automated walking, it is very likely that the strength of the ankle motors will need to be increased. Unfortunately, this will raise the weight of the leg.

9.6 Grasping

The team designed and assembled actuated grasping hands for Koalby based on the Dynamixel XL-320. The hands can be easily attached to the forearm, allowing for interchangeability with the existing parts. These hands were not attached or tested on Koalby due to time constraints. While one initial goal of the project was to grasp objects with actuated hands, this goal will instead be passed on to a future project, with the team's existing hand design available as a starting point.

9.7 Demonstration at TouchTomorrow 2022

The WPI TouchTomorrow event was the first opportunity for the team to demonstrate their project (see Figure 9.7.1). Overall the event was a big success. The robot was able to perform numerous different actions including dances like the disco and macarena, interactions like waving and shaking hands, and even a marching motion (see Appendix H). These actions were able to run continuously as a constant stream of people wandered by and interacted with the

team. The batteries proved to be an effective solution to powering the robot without a tether (section 8.3) and mechanically the robot had no issues.



Figure 9.7.1: Koalby on test stand at TouchTomorrow

More important than all of this though was that the team was able to accomplish their goal of inspiration first hand. Many of the participants of this event were families with young children who would point at the robot because that was the exhibit they wanted to go look at. They would gain a huge smile after shaking Koalby's hand and would dance with him as he did the disco. Some would ask how the team accomplished the project and they could share about 3D printing and the ability to record and replay the motions. It was clear that Koalby had a huge positive impact that day on the people who visited. The capability of a small-scale humanoid robot to inspire and excite people, especially young children was demonstrated to its full extent, as shown in Figure 9.7.2 below.



Figure 9.7.2: Children watching Koalby at TouchTomorrow

9.8 Demonstration at Project Presentation Day

The team demonstrated the project at WPI's Project Presentation Day. Similarly to the TouchTomorrow event, Project Presentation Day proved successful for the team. This event allowed the team to demonstrate and present their project at a higher level, speaking to professors and engineering professionals. Koalby had no mechanical, software, or electrical failures throughout the day and the team won first place in the Department of Mechanical and Materials Engineering division.



Figure 9.8.1: Robotics Engineering Project Presentation Day



Figure 9.8.2: Awarded 1st Place in Mechanical and Materials Engineering

10 Conclusions

At the end of this project, the team was successfully able to reproduce the functionalities demonstrated in the original Poppy Project robot, and was able to achieve the other predetermined objectives outlined at the beginning of the project timeline.

Beyond recreating the functionalities of the Poppy robot, the team's first objective was to find alternate components to replace pieces of the Poppy design that would ultimately reduce the cost of creation. The team was successfully able to achieve this goal with a cost reduction of approximately \$6,000 - 3,000, with the original Poppy design costing ~\$10,000 - \$7,000, depending on whether parts were obtained through the Poppy group or through external means, and the new Koalby design costing ~\$4,000.

In addition, the team's second and tertiary goals of giving Koalby the ability to walk, and to be able to grasp objects were both partially achieved. The team was successfully able to untether the robot by supplying fully onboard battery power and wifi capabilities for remote operation which was the first major step towards achieving walking. Koalby also achieved the ability to walk short distances with user assistance, and is planned to walk independently through the work of future projects. Finally, actuated grasping hands were fully designed and assembled for Koalby's use, however, were not fully implemented due to time constraints. These graspers could easily be implemented through future work as well.

10.1 Broader Impacts

10.1.1 Engineering Ethics

The applications of this project include making a cost accessible humanoid robot for the purpose of human kinematic study, human robot interaction, education, and inspiration. These applications promote the enhancement of human welfare. For one, improving knowledge about human kinematics can help the medical field, specifically comparing the motion of limbs under normal and abnormal conditions (An & Chao, 1984). Creating a platform to study human robot interaction, especially with a humanoid robot, allows for further insight into how humans react to their robotic counterparts, and specifically how youth can interact with these types of robots. Lastly, this project promotes the welfare of children through education and inspiration. This project could be used to teach students about how to create a humanoid robot, and the fun movements of the robot can create awe within children and inspire them to go into STEM fields.

The methods and results of this project were reported honestly and impartially. All steps and data are accurately portrayed in this report. Additionally, this project is an advanced creation of a low-cost humanoid robot. With 25 degrees of freedom and motion nearly identical to that of a human, this project increases the prestige of the engineering profession.

10.1.2 Societal Impact

The completion of this project and its successful achievement of cost reduction could lead towards larger societal impacts. Firstly, the large reduction in cost will make Koalby's design more accessible to a wider number of researchers. In the spirit of the original open source Poppy design, Koalby's mechanical, electrical, and software materials will be accessible online. With the lower price point as well as an accessible manufacturing method of 3D printing, Koalby should be more reasonable to reproduce to continue research into human robot interaction (HRI), human kinematic studies, and to use in educational fields. In the areas of education and inspiration, this design may also be able to be scaled down further to reduce the cost even further, allowing schools to use the design as an educational model and to teach complex robotic theories such as kinematics and advanced control algorithms.

10.2 Future work

Despite the team's success this year, there are a number of areas where Koalby can be improved by future groups. These areas include hardware (design simplification, motor selection, implementing grasping hands) and software (sensors and control code).

10.2.1 Hardware

The robot's hardware is sufficient as a proof of concept, but it has a number of areas that can be improved. These include simplifying the design to reduce the number of 3D printed parts and screws needed to assemble the robot, selecting motors more carefully at each joint to maximize torque where needed, and fully implementing grasping hands to allow Koalby to manipulate objects.

10.2.1.1 Design Simplification

In order to quickly modify the Poppy design to support less expensive HerkuleX motors, the team developed an adaptor (as shown in Figure 10.2.1) that includes both MX-28 and HerkuleX holes. This can be connected to a HerkuleX motor and slotted directly into an existing Poppy part. This adaptor reduced the design work required to build Koalby by a significant amount.

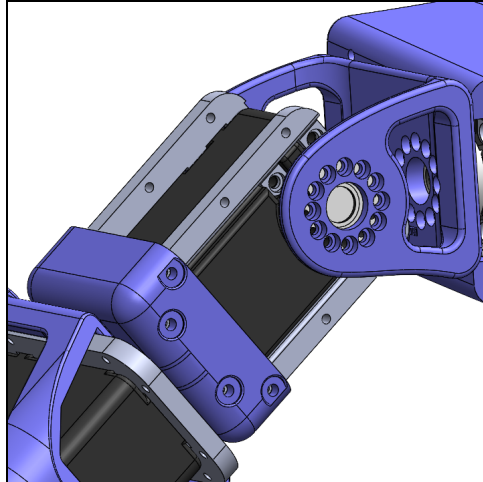


Figure 10.2.1: Adaptor Pieces (gray) connect a HerkuleX to Poppy hole spacing

The disadvantage of this adaptor is that it significantly increases the number of printed parts and screws used on Koalby. Each adaptor needs 2 sets of screws, one set to connect it to the motor and another set to connect it to the printed part. Future work could improve the design by including the correct mounting holes for the new HerkuleX motors directly on the larger parts instead of using adaptors. This change would help to reduce cost, weight, print time, and assembly time.

Another design simplification that could be investigated is an easier battery removal system. Koalby uses onboard batteries instead of external power. These include two large batteries in the leg to power the HerkuleX motors, and a smaller battery to power the Dynamixels which is located in the head. Each battery is accessed by removing a set of 4 screws. Figure 10.2.2a shows the screws securing the shin battery, and figure 10.2.2b shows the head lid screws.

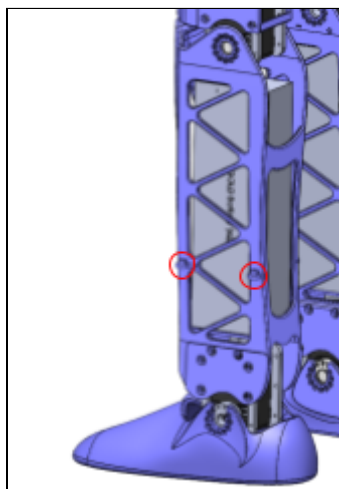


Figure 10.2.2a: Modified shin with battery

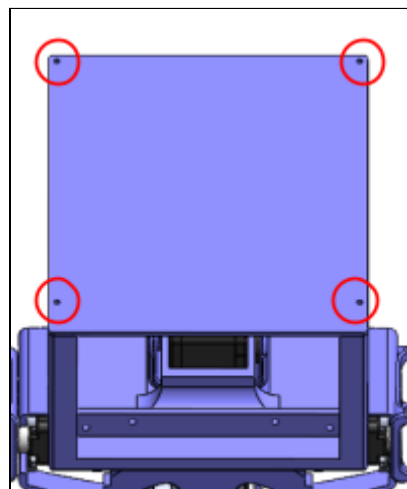










Figure 10.2.2b: Head with lid

Because these screws must be undone to remove the batteries, accessing them is difficult to do on short notice. To improve this, the screw based connector could be replaced with some sort of snap fit that could make the batteries accessible without tools. With a battery life of only 4 hours, the ability to rapidly swap batteries would improve the ability of the robot to be used for prolonged testing/demonstrations.

10.2.1.2 Motor Selection

The team's motor selection this year was based on direct substitution of HerkuleX motors for MX-28's, leaving the design otherwise unchanged. However, Dynamixel actuators are available in many sizes. Especially relevant are the newer X-series motors (<https://www.robotis.us/x-series/>), which were developed after the Poppy Project. These motors can be purchased from the same source, and use the same communication protocol, so the difficulty of acquiring and using more types of actuators should be negligible. Table 10.2.1 below shows a section of the selection guide, illustrating the number of options available.

Table 10.2.1: Dynamixel Selection guide (reproduced as is from http://en.robotis.com/service/selection_guide.php)

ITEM	Series	Input Voltage			Performance Characteristics				Resolution			Dimensions (W x H x D) [mm]	Weight [g]
		Min. [V]	Recommended [V]	Max. [V]	Voltage [V]	Stall Torque [N-m]	Stall Current [A]	No Load Speed [rpm]	Resolution [deg/pulse]	Step [pulse/rev]	Angle [degree]		
 902-0127-000 XH430-W210-T e-Manual	DYNAMIXEL X	10.0	12.0	14.8	12.0	2.50	1.3	50.0	0.0879	4	360	28.5 X 46.5 X 34.0	82.00
 902-0121-000 XH430-W210-R e-Manual	DYNAMIXEL X	10.0	12.0	14.8	12.0	2.50	1.3	50.0	0.0879	4	360	28.5 X 46.5 X 34.0	82.00
 902-0096-000 MX-28AT e-Manual	DYNAMIXEL	10.0	12.0	14.8	12.0	2.50	1.4	55.0	0.0879	4	360	35.6 X 50.6 X 35.5	77.00
 902-0095-000 MX-28AR e-Manual	DYNAMIXEL	10.0	12.0	14.8	12.0	2.50	1.4	55.0	0.0879	4	360	35.6 X 50.6 X 35.5	77.00
 902-0057-000 MX-28T e-Manual	DYNAMIXEL	10.0	12.0	14.8	12.0	2.50	1.4	55.0	0.0879	4	360	35.6 X 50.6 X 35.5	72.00
 902-0054-000 MX-28R e-Manual	DYNAMIXEL	10.0	12.0	14.8	12.0	2.50	1.4	55.0	0.0879	4	360	35.6 X 50.6 X 35.5	72.00
 902-0175-000 XV430-T200-R e-Manual	DYNAMIXEL X	10.0	12.0	14.8	12.0	2.30	1.3	53.0	0.0879	4	360	28.5 X 46.5 X 34.0	96.00
 902-0147-000 XC430-W240-T e-Manual	DYNAMIXEL X	10.0	12.0	14.8	12.0	1.90	1.4	70.0	0.0879	4	360	28.5 X 46.5 X 34.0	65.00

With these options, a future team could spend more time on the motor selection phase of the project, and calculate or simulate the torque required at each joint to select the correct motor. Some joints, like the leg joints, seem to have difficulty moving the heavy batteries and could

benefit from a higher torque actuator. Other joints are unlikely to need the full torque of their motor, and these could be scaled down to reduce cost.

Specific Dynamixel motors of note are the dual axis 2XC-430 (robotis.us/dynamixel-2xc430-w250-t/) and smaller 2XL-430 (robotis.us/dynamixel-2xl430-w250-t/). These motors contain 2 Dynamixels packaged in a single casing, providing two degrees of freedom with 1 motor. The larger 2XC-430 motors are similar in torque to the HerkuleX motors, while the smaller 2XC-430 could potentially replace the head AX-12s. Figure 10.2.3 below shows these theoretical locations.

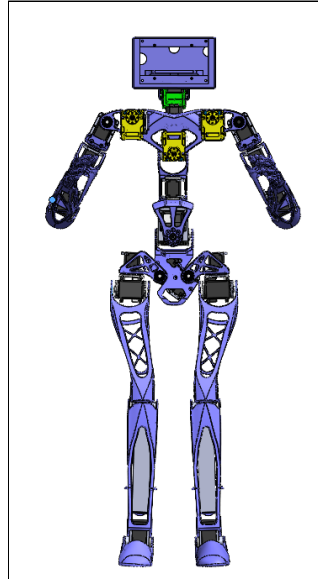


Figure 10.2.3: Dual axis motor locations (2XC-430 yellow, 2XL-430 green)

The primary advantage of these motors would be reducing complexity. Joints like the shoulder and double rotation use a number of printed parts and adaptors to get 2 axes of motion. Replacing these pairs of motors with a single motor would reduce the number of printed parts required, helping to simplify the design. Table 10.2.2 below shows the comparison between the HerkuleX motor and the Dynamixel 2XC-430.

Table 10.2.2: Comparing HerkuleX and Dynamixel 2XC-430 motors

Motor	Torque	Speed	Weight	Cost
HerkuleX 0201	2.35 Nm	68 RPM	120 g (for 2 motors)	\$264 (for 2 motors)
Dynamixel 2XC-430	1.8 Nm	57 RPM	102 g	\$239.9

This table illustrates the differences between these motors. The 2XC-430 has only 80% of the torque of the HerkuleX 0201, but is otherwise similar in price, weight, and speed. If simulations showed this torque loss was an acceptable tradeoff, then this motor could significantly simplify the design.

10.2.1.3 Grasping hands

One of the goals for this project was to add grasping hands in place of Poppy's humanoid hands. The team designed and printed a prototype based on a hand design for Poppy made by a community member (Figure 10.2.4). This hand uses Dynamixel XL-320 (<https://www.robotis.us/dynamixel-xl-320/>) actuators and was assembled, but the team was unable to test it due to time constraints. Future work would include attaching it to Koalby, testing it, and perhaps adding more degrees of freedom if desired.

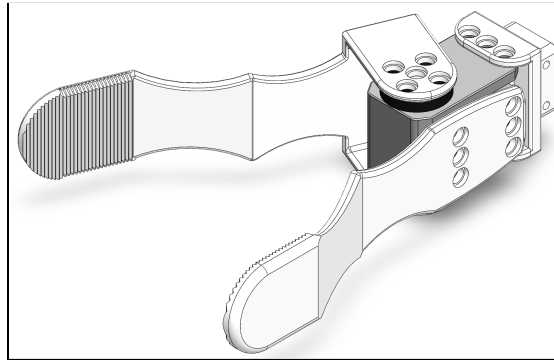


Figure 10.2.4: Grasping hand design

10.2.2 Software

Future software changes for Koalby can focus on the high-level control, modifying the Arduino firmware if needed to enable additional features.

10.2.2.1 Sensing

Koalby currently features no sensors outside of those integrated into the motors. Implementing more advanced features will require more sensing power. Potential options from the Poppy project include IMU's, cameras, and pressure sensors.

An IMU could allow Koalby to measure its orientation relative to gravity, which could greatly enable self-balancing, walking, and motions that require Koalby to bend at the waist. Cameras could help allow object detection once grasping is added, object avoidance for walking, or even motion detection and replication for Koalby to mirror what someone is doing in front of it. Pressure sensors would lead the way for precise grasping of objects or detecting when to fully put weight on a foot while walking.

10.2.2.2 Kinematics

Implementing kinematics was initially something that this team had hoped to accomplish. While the overall goals were still able to be accomplished without kinematics, there are still many benefits to continuing the work to build this into the Koalby system. Kinematics allow for more precise and controlled motions rather than just recording and replaying them. It also allows for the robot to actually know where it is functioning in space. This is important for developing motions that avoid hitting obstacles or itself. Implementing kinematics can also help smooth out

motions and make Koalby look less robotic and more humanoid. More complex motions that future teams would look to implement - such as walking - depend on the precise control that kinematics allows.

10.2.2.3 Self-Balancing

Currently Koalby can stand upright without any assistance. However, nothing in the code prevents Koalby from unbalancing itself and falling, or ensures that it starts in a balanced state. This means that Koalby needs to operate almost entirely within the constraints of a supporting stand. Koalby can attempt to walk, but is nearly guaranteed to fall without support for a stand or human correction. One of the goals for this project was to move to battery power to allow for Koalby to walk independently of a tether. With this goal accomplished, Koalby primarily needs development of balancing walking algorithms integrated with sensor feedback to allow for balancing to be accomplished. The goals of the current project and the route development took was chosen with future applications of self-balancing and walking in mind. While the system is not currently ready to accomplish this task, it should be feasible for a future group to allow for Koalby to balance on its own.

10.3 Project Experience

The two most important skills the team utilized to successfully complete Koalby were good time and team management as well as keeping everything organized. Since MQP has the requirement to work on the project for 15-17 hours per week, delegating tasks, asking others for help when needed, and consistently reporting progress were key factors in the success of the team. Additionally, each member of the team brought different skills and knowledge to the project. The members of the team can be contacted at the email addresses provided in Appendix I for any inquiries about the project.

Alexandria Lehman

The main skill I learned at WPI and applied to this project is my knowledge in Solidworks. I learned Solidworks in ES 1310 (Introduction to Computer Aided Design). This class taught me how to use the different tools in Solidworks, and what is possible to do. The majority of my contribution to this project was modifying the Poppy robot to match our goals, such as modifying joints to match the HerkuleX motors, or lengthening and splitting the shin for the batteries. The class also taught me about Solidworks assemblies. I created a full robot assembly and different body part subassemblies, which allowed us to see the robot in its entirety. It also allowed me to check for interferences between systems. I was also able to apply soft skills that I have learned over the years at WPI. Specifically, working on a team and how to divide up tasks based on everyone's strengths. This has been very useful in this project as it has made it so everyone has a task to work on, increasing the efficiency of the team.

This project also taught me various technical skills I can apply to future projects. Over the last year, I learned a lot about resin 3D printing, a technology I did not know existed before the project. After 3D printing the entirety of the robot using resin printing, I now know how resin printing works and how to print with a resin printer, including how to design for this type of manufacturing.

One thing I wish I knew at the beginning of the project is when to stop trying one solution and move onto another. I spent a majority of B term working on modifying and splitting the thigh part since it was too tall to print on the resin printer we had at the time. The problem is that the thigh part had a bend feature in it, meaning it could not be cut in half on a flat plane as the cut feature would be bent. In the end, we decided to get a larger printer to allow for the thigh. Had I realized earlier that splitting the thigh would not work, I would have been able to allocate my time to other tasks that ended up falling behind schedule, such as assembling the legs of the robot and designing and prototyping hands.

Anthony Galgano

This project gave me the opportunity to work in a team to solve problems and deliver a final product. Overall, I used my knowledge and skills of time management and working as a team so that each person on the team was working effectively. In my previous coursework, I learned a lot about motors in RBE 1001 and battery powered systems in ECE 2799. ECE 2799 taught me about different types of batteries and how to calculate loads from a system. This was helpful when deciding what size batteries we would be using. Additionally, RBE 1001 taught me about basic motor selection and the different abilities and constraints of motors. This was helpful when the team was researching new motors to replace the Dynamixel MX-28AT motors.

While I was using my existing skills to work on Koalby, I also learned many new skills. Since we switched to DLP printing, I learned how resin printing works and the strengths and weaknesses of it. Before this year, I never knew what resin printing was and now I have realized how accurate and precise it can print parts especially with no visible layer lines. Another skill I learned was how to use multi-threading in Python. Before this project I had never needed to write a multi-threading program so it was interesting to learn the capabilities of multi-threading to create more powerful scripts in Python.

One key thing I wish I knew before the project is that resin printing is far better than FDM printing. The team spent about 3-4 weeks trying to FDM print parts which was a waste of time in the end because FDM could not give us the resolution we needed. Another realization I wish the team had earlier is that the Poppy Project ended in 2014, meaning that many aspects about it were outdated, and software platforms have changed significantly. For example, when trying to use the Poppy Projects original code base, the team used 2-3 weeks to read through it and was not able to use it. Overall, this project experience was very beneficial to me and my career and I enjoyed working on Koalby.

David Fournet

During this project, I was able to put many of the skills I had learned in previous coursework and projects at WPI to use. The first of these skills was the use of DLP resin 3D printing, which I had been aware of for a number of years but finally learned to use during the summer prior to this project through my work on a partial hand prosthetic team at WPI. I was also able to put my skills in code structure, data types, and programming into use on this project which I have learned and honed during my time at WPI through courses such as software engineering, object-oriented programming, and the robotics course series. Finally, I felt that my knowledge of github and version control was very useful during this project to keep our team's work clean, well documented, and safe from coding mishaps.

One new skill that I got the privilege of encountering while working on Koalby was the use of multithreading in Python. I had not previously used multithreading in the past and I enjoyed learning how to use its functionalities to bolster our capabilities in Koalby's codebase. I also enjoyed working with and getting to explore more of the capabilities of DLP printing and ways to ensure successful, clean prints.

Something that I wish I had known at the beginning of this project would be that our team was going to need to move over to building our own code library from scratch rather than use the poppy project library. Due to poor documentation and outdated libraries and functionalities, the original poppy library simply did not apply well for our goals. If we had known this going into the project, we could have devoted more time towards more functionalities, such as auto balancing algorithms, rather than time spent reviewing and understanding the original library's logic.

Overall, I very much enjoyed my time working on this project and feel incredibly lucky to have gotten to work with everyone on this team. Our group worked well together, delegated tasks easily and fairly, and became good friends over the course of this year. I cannot wait to see what everyone does going forward and am excited to see where Koalby ends up through the work of future MQPs.

Raymond Beazley

This project allowed me to use skills I had already learned and also develop my skills further while bringing the project to its final state. In the project I worked on many different aspects including initial design, code development, assembly, testing, and documentation. I started by using my knowledge in Solidworks, which had been developed through many other projects and classes at WPI, to design the motor adaptor and spacer which were able to be used through nearly all subsystems of the robot. I then continued to use Solidworks to help redesign all the segments of the arms and torso to work with the new adaptor.

At this point I stepped away from the CAD when developing the code became a key priority. Given my experience coding in the Robotics curriculum I was able to transition quickly to diving into the Poppy project code base (written in Python) to better understand how it worked and whether it was feasible to adapt it for our purposes. After we decided that the scope of the

entire Poppy project was both too broad and complicated for the goals of our project I worked on developing our own kinematics using the same library as the Poppy project. I was able to get this running on our system to the point of locating and diagnosing the key bug that would prevent future progress. With this bug understood, we decided to pause progress on kinematics to pursue more pressing goals of the project, and I shifted towards assisting with testing and debugging. After the system was complete I wrote documentation for the project, including the assembly instructions which will allow for future groups to recreate our project.

This project may have not taught me any new skills that can be broadly categorized, but it did certainly teach me a lot. I was able to develop my skills and comfort using Solidworks and writing/understanding code in Python. I was able to experience a year-long group project and learn how to communicate, plan, and complete such a large scale project when compared with all my previous project experience. I was able to learn about methods of control like the primitive manager system which could be used effectively in other robotic applications. I grew as a student, an engineer, and a person through this project.

William Engdahl

In this project I used and further many of the skills I learned at WPI both in and out of classes. In this project, I worked primarily on motor control and communication. This involved programming Koalby's Arduino firmware, developing the USB communication protocol, and understanding the operation of HerkuleX and Dynamixel motors. My previous robotics coursework gave me significant experience using and programming Arduinos which I was able to use when programming Koalby.

One of the areas I focused on was testing and understanding the HerkuleX and Dynamixel motors. In order to program the low-level motor control, I spent significant amounts of time reading the motor datasheets and library documentation to figure out the exact mechanics of these motors. In order to test this behavior, I wrote a number of testing programs separate from the main firmware which I could upload to verify motor behavior or assist in setting up firmware.

Throughout the project, I did not so much learn new skills as much as applied the skills I had learned previously. In addition to working on the Python code, I also helped with CAD, documentation, and testing Python code. The biggest new skill I gained was familiarity with the resin printing process. My prior experience with 3D printing involved FDM machines, and the team's resin printer used a dramatically different process. Overall, I gained a lot of practical experience during the course of this project and hope that future groups are able to build on the work we have completed this year.

11. References

- An, K. N., & Chao, E. Y. (1984). Kinematic analysis of Human Movement. *Annals of Biomedical Engineering*, 12(6), 585–597. <https://doi.org/10.1007/bf02371451>
- Casley, S. V., Choopojcharoen, T., Jardim, A. S., & Ozgoren, D. B. (2014). (rep.). *IRIS Hand: Smart Robotic Prosthesis*. Retrieved January 13, 2022, from <https://digital.wpi.edu/pdfviewer/n296x092x>.
- Discover the poppy open-source technology created at INRIA*. Génération Robots. (n.d.). Retrieved October 13, 2021, from <https://www.generationrobots.com/en/312-poppy-humanoid-robot>.
- Dynamixel MX-28AT*. ROBOTIS. (n.d.). Retrieved January 13, 2022, from <https://www.robotis.us/dynamixel-mx-28at/>
- Hard-tough resin __blue: Esun 3D printing materials*. Hard-Tough Resin __Blue | eSUN 3D Printing Materials. (n.d.). Retrieved April 13, 2022, from <https://www.esun3d.net/products/242.html>
- Heney, P. (2018, August 24). *igus Delta Robot a Low-Cost Option for Assembly Tasks*. The Robot Report. Retrieved January 13, 2022, from <https://www.therobotreport.com/igus-delta-robot-cost-effective-assembly/#:~:text=The%20igus%20delta%20robot%20costs,controller%20package%2C%20according%20to%20Mowry>.
- High temp resin*. <https://www.esun3d.com/>. (n.d.). Retrieved April 13, 2022, from <https://www.esun3d.com/high-temp-resin-product/>
- HS-7954SH High Voltage Ultra Torque Servo*. Hitec RCD Inc. HS-7954SH High Voltage Ultra Torque Servo | Horizon Hobby. (n.d.). Retrieved January 13, 2022, from https://www.horizonhobby.com/product/hs-7954sh-high-voltage-ultra-torque-servo/HRC37954S.html?gclid=Cj0KCCQiA47GNBhDrARIsAKfZ2rDIWkVdMoKgqeWTSYotWNTIe5I6XQjp8VnISDmjpOWvDR0IM4oVBjEaAIKbEALw_wcB
- The HUBO 2 humanoid robot is priced at \$400,000*. Luxatic. (2017, August 31). Retrieved January 13, 2022, from <https://luxatic.com/the-hubo-2-humanoid-robot-is-priced-at-400000/#:~:text=Tech%20%26%20Leisure%20%E2%80%A3-.The%20HUBO%20%20Humanoid%20Robot%20is%20priced%20at%20%24400%2C000.at%20the%20speed%20of%20light>.
- Iglesias, J. G., Bahón Cecilio Angulo, & García Manel Velasco. (2016). *Design, modelling and control of a biped robot platform based on Poppy Project* (thesis). *UPCommons*. Retrieved October 13, 2021, from <https://upcommons.upc.edu/handle/2117/98665>.
- Iverach-Brereton, C., Baltés, J., Anderson, J., Winton, A., & Carrier, D. (2014). Gait design for an ice skating humanoid robot. *Robotics and Autonomous Systems*, 62(3), 306–318. <https://doi.org/10.1016/j.robot.2013.09.016>
- Johansson, B., Tjøstheim, T. A., & Balkenius, C. (2020). Epi: An open humanoid platform


- for Developmental Robotics. *International Journal of Advanced Robotic Systems*, 17(2), 172988142091149. <https://doi.org/10.1177/1729881420911498>
- Kerzel, M., Strahl, E., Magg, S., Navarro-Guerrero, N., Heinrich, S., & Wermter, S. (2017). Nico — neuro-inspired Companion: A developmental humanoid robot platform for multimodal interaction. *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 113–120. <https://doi.org/10.1109/roman.2017.8172289>
- Know Your Materials: SLA Tough Resin: Fast Radius*. Know Your Materials: SLA Tough Resin | Fast Radius. (2022, April 6). Retrieved April 13, 2022, from <https://www.fastradius.com/resources/know-your-materials-sla-tough-resin/#:~:text=eSun%20Hard%2DTough%20Resin&text=The%20best%20wavelength%20is%20between,flexural%20strength%3A%2070%2D80%20MPa>
- Langnau, L. (n.d.). *How tensile strength relates to 3D printing*. Make Parts Fast. Retrieved April 14, 2022, from [https://www.makepartsfast.com/how-tensile-strength-relates-to-3d-printing/#:~:text=PLA%20\(polylactic%20acid\)%2C%20with,degrade%20when%20exposed%20to%20light](https://www.makepartsfast.com/how-tensile-strength-relates-to-3d-printing/#:~:text=PLA%20(polylactic%20acid)%2C%20with,degrade%20when%20exposed%20to%20light)
- Lapeyre, M., N'Guyen, S., Le Falher, A., & Oudeyer, P.-Y. (2014a). Rapid morphological exploration with the poppy humanoid platform. *2014 IEEE-RAS International Conference on Humanoid Robots*, 959–966. <https://doi.org/10.1109/humanoids.2014.7041479>
- Lapeyre, M. (2014b, March 8). *[assembly instructions] motor naming convention and Configuration*. Poppy Forum. Retrieved November 29, 2021, from <https://poppy.discourse.group/t/assembly-instructions-motor-naming-convention-and-configuration/87>.
- Lapeyre, M., Rouanet, P., & Oudeyer, P.-Y. (2013). The poppy humanoid robot: Leg design for Biped Locomotion. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. <https://doi.org/10.1109/iros.2013.6696375>
- Lapeyre, M., Rouanet, P., & Grizou, J. (n.d.). *Poppy humanoid: Advanced and easy to use open source humanoid robot*. Poppy Project - Poppy Humanoid. Retrieved October 28, 2021, from <https://www.poppy-project.org/en/robots/poppy-humanoid/>.
- Larrier, M. A., Saint-Elme, E., Kracinovich, C., & Renshaw, D. (2017). (rep.). *Accurate Prosthetic Hand*. Retrieved January 13, 2022, from <https://digital.wpi.edu/pdfviewer/1g05fd479>.
- Maggiali, M., Fiorio, L., Pattacini, U., & Derito, A. (2019). (rep.). *Technical Specifications of the iCub Platform*. Retrieved January 13, 2022, from https://www.iit.it/documents/175012/528824/Technical-specs_iCub_robot_Rev_2.3_05082019+%281%29.pdf/44d1e600-7a09-12e0-5eeb-5e3a2444e78c.
- Moreno, P., Nunes, R., Figueiredo, R., Ferreira, R., Bernardino, A., Santos-Victor, J., Beira,

- R., Vargas, L., Aragão, D., & Aragão, M. (2015). Vizzy: A humanoid on Wheels for Assistive Robotics. *Advances in Intelligent Systems and Computing*, 17–28.
https://doi.org/10.1007/978-3-319-27146-0_2
- Nao Robot V6*. Eduporium. (n.d.). Retrieved January 13, 2022, from <https://www.eduporium.com/nao-v6-robot.html>
- PETG. Curbell Plastics. (n.d.). Retrieved April 14, 2022, from <https://www.curbellplastics.com/Research-Solutions/Materials/PETG>
- Popov, D., Klimchik, A., & Afanasyev, I. (2017). Design and stiffness analysis of 12 DOF poppy-inspired humanoid. *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*, 2, 66–78.
<https://doi.org/10.5220/0006432000660078>
- Poppy humanoid: Advanced and easy to use open source humanoid robot*. Poppy Project - Poppy Humanoid. (n.d.). Retrieved October 13, 2021, from <https://www.poppy-project.org/en/robots/poppy-humanoid/>.
- Prasanna, V., & Ashok, G. (2021). Design, Modeling And Control of A Biped Robot Platform Based On Poppy Project. *International Journal of Scientific Engineering and Technology Research*, 10, 7–11. Retrieved October 13, 2021, from <http://ijsetr.com/uploads/426315IJSETR17692-02.pdf>.
- Programmable Humanoid Robot NAO V6*. Génération Robots. (n.d.). Retrieved January 13, 2022, from <https://www.generationrobots.com/en/403100-programmable-humanoid-robot-nao-v6.html#:~:text=Programmable%20humanoid%20NAO6%20specifications%3A,90%20minutes%20in%20normal%20use>
- RH2D Advanced manipulator*. Seed Robotics. (n.d.). Retrieved October 13, 2021, from <https://www.seedrobotics.com/rh2d-advanced-manipulator>.
- Roberts, E. (1999). *Robotics: A Brief History*. Robotics: A brief history. Retrieved October 13, 2021, from <https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html#:~:text=The%20earliest%20robots%20as%20we,industry%2C%20but%20did%20not%20succeed.>
- Ruko 6088 programmable robot, gesture sensing intelligent remote control robot for kids 3-8years, Christmas birthday gift*. RuKo. (n.d.). Retrieved January 13, 2022, from <https://www.ruko.net/products/ruko-6088-programmable-robot>
- Saeedvand, S., Jafari, M., Aghdasi, H. S., & Baltes, J. (2019). A comprehensive survey on humanoid robot development. *The Knowledge Engineering Review*, 34.
<https://doi.org/10.1017/s0269888919000158>
- The story behind the Poppy Project*. Poppy Project - About. (n.d.). Retrieved October 13, 2021, from <https://www.poppy-project.org/en/about/>.
- What happened to the Honda Robot?* TouchUpDirect. (2022, January 3). Retrieved January

- 13, 2022, from
<https://touchupdirect.com/blog/what-happened-to-the-honda-robot/#:~:text=Honda%27s%20stated%20intent%20of%20making,hiring%20a%20real%20human%20assistant.>
- Wikimedia Foundation. (2021, December 4). *Robotics*. Wikipedia. Retrieved December 14, 2021, from <https://en.wikipedia.org/wiki/Robot>.
- Wikimedia Foundation. (2021, October 11). *Robot*. Wikipedia. Retrieved October 13, 2021, from <https://en.wikipedia.org/wiki/Robot>.

12. Appendices

Appendix A: 3D printing (resin) Guide

 Resin Printing Instructions

<https://docs.google.com/document/d/1UpJHPfzRCEARCTsiDu4Fe1rfUhEgAGO7IlvqrENxuvU/edit>

Appendix B: Cost Breakdown Between Poppy and Koalby

Component	Unit cost	Quantity (Poppy)	Cost (Poppy)	Quantity (Koalby)	Cost (Koalby)
3D Printed Parts (kilograms of resin)	\$73.56	1	\$73.56	1	\$73.56
Motors					
Dynamixel MX-28AT Motors	\$260.00	19	\$4,940.00	-	-
Dynamixel MX-64AT Motors	\$340.00	4	\$1,360.00	4	\$1,360.00
Dynamixel AX-12A Motors	\$49.00	2	\$98.00	2	\$98.00
HerkuleX DRS-0201	\$132.00	-	-	19	\$2,508.00
Servo Horns					
HN07-N101 (MX-28 Servo Horn)	\$9.70	19	\$184.30		\$0.00
HN07-I101 (MX-28 Idler Horn)	\$15.40	9	\$138.60		\$0.00
HN05-N102 (MX-64 Servo Horn)	\$11.90	4	\$47.60		\$0.00
Custom Dynamixel Horn MX-64, set of 14 (Ordered from SendCutSend)			-		\$30.00
Hardware					
M3 Socket head screws	\$0.06	300	\$18.00	300	\$18.00
Electronics					
Raspberry Pi 4	\$35.00	1	\$35.00	1	\$35.00
Arduino Mega Clone	\$20.00		\$0.00	1	\$20.00
Raspberry Pi Screen	\$55.00	1	\$55.00	1	\$55.00
Total Cost:			\$6,950.06		\$4,197.56

Appendix C: Setup of Github/Grabcad/Google Drive

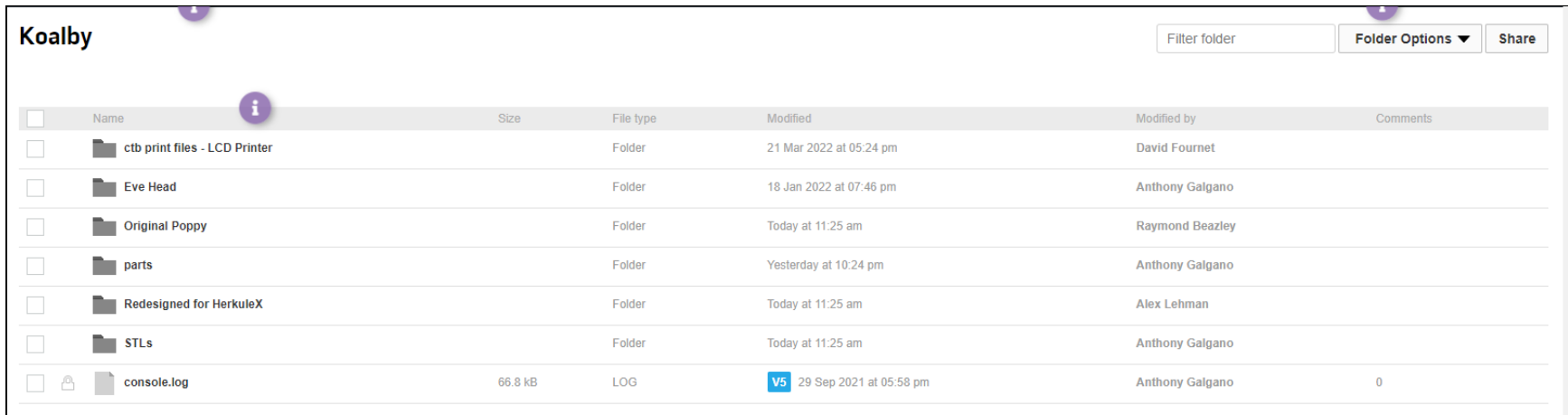
The team stored part models on GrabCad and code on Github

GrabCAD:

The GrabCad repository contains all Solidworks part files and assemblies. This is not publicly available online, access can be requested from any of the team members.

To modify this repository, download GrabCad desktop and set up the project. Instructions can be found here:

help.grabcad.com/article/103-workbench-desktop-app-users-manual



	Name	Size	File type	Modified	Modified by	Comments
<input type="checkbox"/>	ctb print files - LCD Printer		Folder	21 Mar 2022 at 05:24 pm	David Fournet	
<input type="checkbox"/>	Eve Head		Folder	18 Jan 2022 at 07:46 pm	Anthony Galgano	
<input type="checkbox"/>	Original Poppy		Folder	Today at 11:25 am	Raymond Beazley	
<input type="checkbox"/>	parts		Folder	Yesterday at 10:24 pm	Anthony Galgano	
<input type="checkbox"/>	Redesigned for HerkuleX		Folder	Today at 11:25 am	Alex Lehman	
<input type="checkbox"/>	STLs		Folder	Today at 11:25 am	Anthony Galgano	
<input type="checkbox"/>	console.log	66.8 kB	LOG	29 Sep 2021 at 05:58 pm	Anthony Galgano	0

As shown in the image above, there are a number of folders within the repository:

- CTB print files - LCD printer
 - Sliced print files for the LCD printer
- Eve Head
 - Parts for Poppy's Eve head design. These are not used on Koalby

- Original Poppy
 - Poppy Project model, used as reference
- Parts
 - Models for 2-axis Dynamixel motors
- Redesigned for HerkuleX
 - Main folder, contains Solidworks models for Koalby and its parts
- STLs
 - STL files for Koalby parts

Github:

<https://github.com/Poppy-MQP>

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


- Kolby-Humanoid
 - Rewritten Python code to control the robot, interfaces with firmware
- 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
- Torso-Poppy
 - Legacy code: cloned from Poppy repository, not currently in use

Appendix D: Assembly Instructions

 Assembly Instructions

https://docs.google.com/document/d/1U__MpuTLPrsCEsMbUcS0TH0UsqzTJfTEnFXiUKoMfY/edit?usp=sharing

Appendix E: Raspberry Pi Setup Instructions

 Raspberry Pi Setup Instructions

<https://docs.google.com/document/d/1wLcgCzkZAl0FGVNLbetXEDFZKqFWIZXvcgTQgD8AIDI/edit?usp=sharing>

Appendix F: Shin Torque Calculations and Initial Arm Testing

 Motor Testing Suite

https://docs.google.com/spreadsheets/d/1adM1Q3Fr-XuB6N-HMxxCtSWIMH2_WFpea7laKOW5xFM/edit#gid=1399018677

Appendix G: Cost and Time Breakdown of DLP Resin Printed Parts

List of Printed Parts:
2 Hand
2 Forearm
2 Upper Arm
2 Arm Connector
2 Shoulder
30 Motor adaptor prints
2 Dynamixel idler servo horns
10 HerkuleX idler servo horns
2 Foot
2 Shin
2 Thigh
2 Hip Connector
2 Hip
1 Pelvis
1 spine
1 abdomen
1 torso
1 neck
1 head
Total printed parts: 68

Assuming use of eSun Hard-Tough Resin (Blue):				
Part List	Volume (mL)	Weight (g)	Price of Resin (USD)	Print Time (hh:mm:ss)
Left Arm	109.24	120.2	6.45	8:00:00
Right Arm	109.24	120.2	6.45	8:00:00
30 hercule motor adaptors	81.79	90	4.83	2:00:00
Servo horns + double rotation mounts	46.89	51.6	2.77	2:00:00
Head + lid + neck	246.76	271.4	14.56	9:30:00
Left Leg	224.12	246.5	13.22	9:18:20
Right Leg	224.12	246.5	13.22	9:18:20
pelvis + abdomen + spine + chest	204.48	224.9	12.06	9:45:17
TOTAL:	1246.64	1371.3	73.56	60:00:00

 Printed Parts List, Volume, Costs, and Times

https://docs.google.com/spreadsheets/d/1uizXj_CDjPUXfgwjGYP6_BqxEv3_nt3z98dCcQkBm0/edit?usp=sharing

Appendix H: Koalby Demonstration Videos

Description	Link
TouchTomorrow 2022	https://photos.app.goo.gl/2dwA4Xcs7rhoTkiB7
Project Presentation Day	https://photos.app.goo.gl/acxoQDFuzY5LWkbT9
Final Compilation of Koalby Motions	https://youtu.be/GyYreHS-OyM
Koalby Interview - David Fournet	https://youtu.be/vohqdSPZAiw

Appendix I: Team Contact Information

Team Member	Email
Alexandria Lehman	alexlehman722@gmail.com
Anthony Galgano	anthony.galgano@gmail.com
David Fournet	davfournet@sbcglobal.net
Raymond “Ian” Beazley	raymondibeazley@gmail.com
William Engdahl	wengdahl1@gmail.com