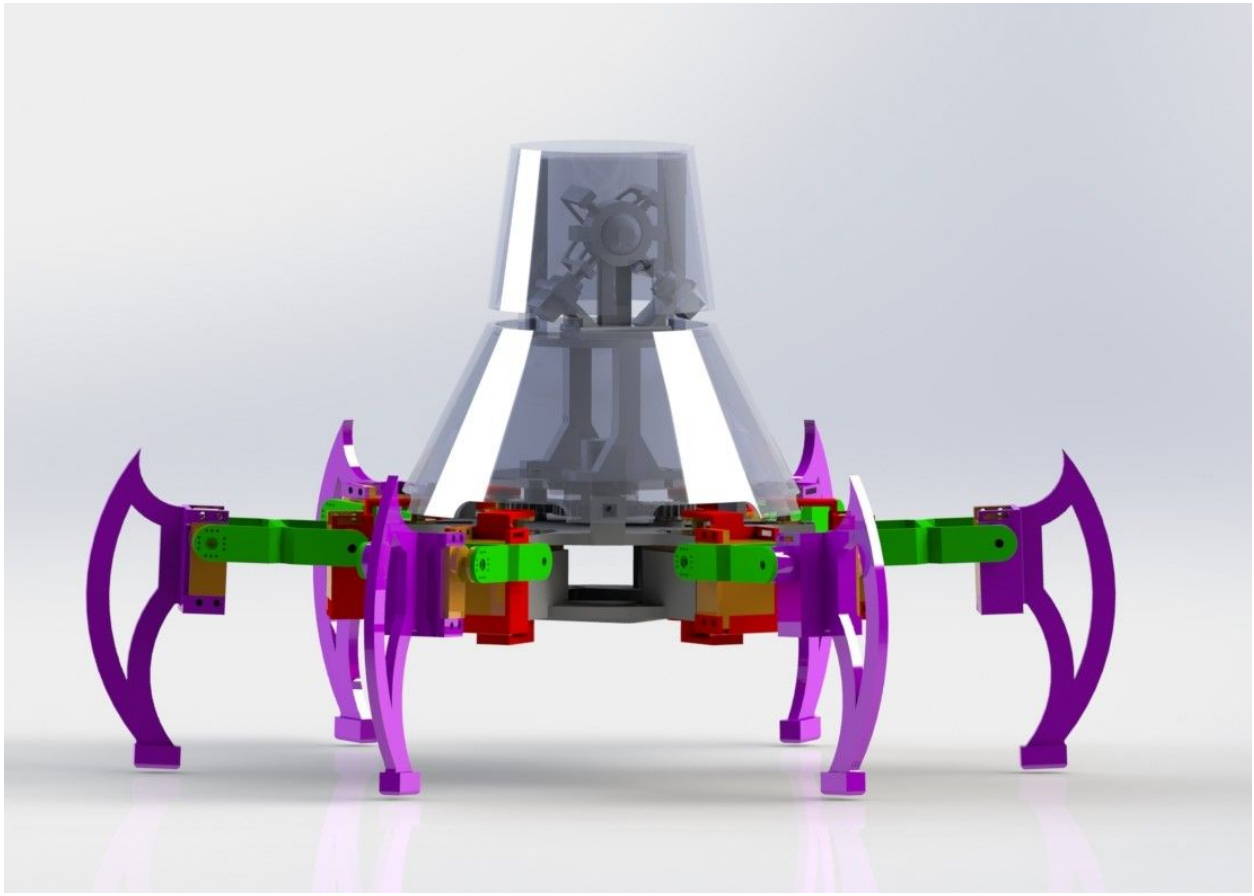
The Guardian

ASME of UCLA: X1 Robotics



Technical Design Report

1 Introduction

1.1 X1 Robotics and the 2018-2019 Team

Founded in 2015, X1 Robotics is one of the University of California, Los Angeles (UCLA) Engineering's student-led project teams with the mission of spreading real world engineering and problem solving skills through the development of challenging robotic systems. Falling under the umbrella organization of the American Society of Mechanical Engineers (ASME) UCLA Student Branch, what sets us apart is our passion and creativity; every year marks the start of a new project decided upon by the diverse team of student engineers, and a new cycle of research, development, and iteration. X1 is proud of the collaborative work environment it fosters and will continue to excel in the field of robotics as it attempts complex, challenging projects pushing UCLA students to their limits.

This year's team consisted of two project leads, three sub-team leads, and 15 sub-team members. Applications were sent out at the beginning of the year, and the team was assembled based on prior knowledge, diversity of experience, and willingness to commit through completion. A full list of members is as follows:

Michael Cui X1 Project Lead *4th Year MechE*	**John Tabakian** X1 Project Lead *2nd Year MechE*

Nicholas Amadeo Mechanical Sub-Team Lead *2nd Year MechE*	**Kenneth Vuong** Mechanical Sub-Team Member *1st Year MechE*
Jennifer Cheng Mechanical Sub-Team Member *4th Year MechE*	**Michael Mitroshin** Mechanical Sub-Team Member *1st Year MechE*
Madeleine Plant Mechanical Sub-Team Member *3rd Year MechE*	**Liam Olive** Mechanical Sub-Team Member *1st Year MechE*

Hayato Kato Controls Sub-Team Lead *2nd Year EE*	**Jingbin Huang** Controls Sub-Team Member *3rd Year EE*
Justin Chang Controls Sub-Team Member *3rd Year EE*	**Olivia Loh** Controls Sub-Team Member *1st Year EE*

Akaash Venkat Software Sub-Team Lead *3rd Year CS*	**Ben Yang** Software Sub-Team Member *3rd Year CS*
Amy Tu Software Sub-Team Member *3rd Year CS*	**Frank Xing** Software Sub-Team Member *1st Year CS*
Anirudh Balasubramaniam Software Sub-Team Member *2nd Year CS*	

1.2 Inspiration

This year's project is inspired by Nintendo Corporation's *The Legend of Zelda: Breath of the Wild*. In the video game, the player encounters an enemy known as a Guardian: a six-legged mechatronic creature with a free-rotating head and laser eye. The creature roams open fields before identifying the player, rapidly approaches them, and then attacks them with its laser. X1's Guardian attempts to mimic its in-game counterpart in aesthetics, functionality, and behavior.

1.3 Design Objectives

Three tiers of success were defined for X1's Guardian. The first tier of objectives revolves around reaching a minimum viable product: the robot must have six legs, an imaging system, and a laser. This would allow it to navigate 3D space, rotate about a fixed point, and perform basic targeting functions. The six legs should be able to move independent of one another—collectively support the weight of the robot—and have enough degrees of freedom to support free movement of the robot. The targeting system must be able to take pictures of the surrounding area and identify a potential target with reasonable accuracy.

The second tier of objectives involved adding basic user input to the functionality of the robot. To complete tier two objects, the robot must receive commands wirelessly and execute them. A simple graphical user interface (GUI) must be completed to allow a user to easily send commands to the Guardian.

The third and final tier of success revolves around autonomous behavior. To complete tier three objects, the robot must be able to operate without user input. It must freely and "organically" navigate a space until a target is located. Upon classifying the target, it must autonomously track and approach the target.

2 System Design

2.1 Mechanical Design Rationale

In the design of the Guardian robot, it was assumed that all parts were to be manufactured via 3D printing technology using polylactic acid (PLA) or acrylonitrile butadiene styrene (ABS) plastic. Due to this chosen manufacturing method, geometric complexity of the desired parts was not a pressing issue that needed to be considered during the design process. However, due to the fact that tapping plastic parts would create weak internal threads that could easily be stripped by metal fasteners, care needed to be taken when designing how the different parts would fasten to each other. The main area where this issue came into consideration was the pin joints connecting the various segments of the leg subassembly to each other. As an example, additional views of Segment 1 of the leg subassembly are shown in Figure 2.1.1.

Figure 2.1.1: *Front and back views of Segment 1 of the leg subassembly*

As stated when previously describing Segment 1 of the leg subassembly, the cylindrical extrusion on the outside of the hollow rectangular section of the segment was designed to hold a 10-32 button head bolt that created a pin joint between Segment 1 and Segment 2. However, due to the fact that the plastic leg segment could not be tapped, to hold the 10-32 bolt in place, a hexagonal hole was designed into the inside of the rectangular section of the segment. During the assembly of the robot, a 10-32 nut was then inserted inside this hexagonal hole, thus allowing the 10-32 bolt to thread into the nut and stay in place. Thus, ultimately, in order to solve the issue of being unable to tap the plastic leg segments, the leg segments were designed such that metal nuts could be used as inserts to create threads in the holes instead. This approach to creating the pin joints was replicated and applied throughout the entire robot in all necessary locations.

In addition to designing the leg and body segments to accommodate the choice of material and manufacturing method, all of the leg and body segments described were also designed to fit the desired servo motors. Whenever the body of a servo motor needed to be mounted onto a part, two to four 4 mm holes were always placed in the part in order to allow the motor to be secured to the part via M4 bolts. Moreover, whenever a part was to be actuated by a servo motor, that part would need to be attached to the horn of the servo motor. Attaching parts to servo horns was facilitated using the following hole pattern:

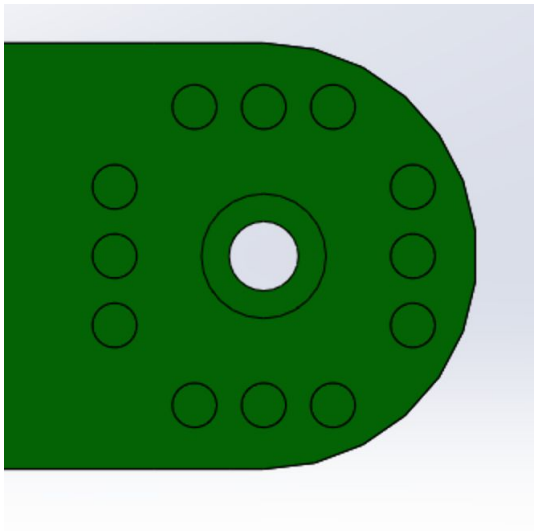


Figure 2.1.2: *Hole pattern used to fix parts to servo horns*

This hole pattern features a central counterbored hole through which an M3 bolt was used to fix the part to the center of the servo horn. However, with just this M3 bolt, if the servo horn were to rotate, the horn would simply slip on the interface of the part. Thus, in order to allow the servo to transmit its torque to the part, M2 screws were inserted through the holes surrounding the counterbored hole and then inserted into similar-sized holes already present in the servo horn. In the final assembly of the robot, only two M2 screws were used per servo motor, but each part was designed with 12 surrounding holes in order to counteract the fact that some of the holes did not line up exactly with the holes in the servo horns.

2.2 Base Subassembly

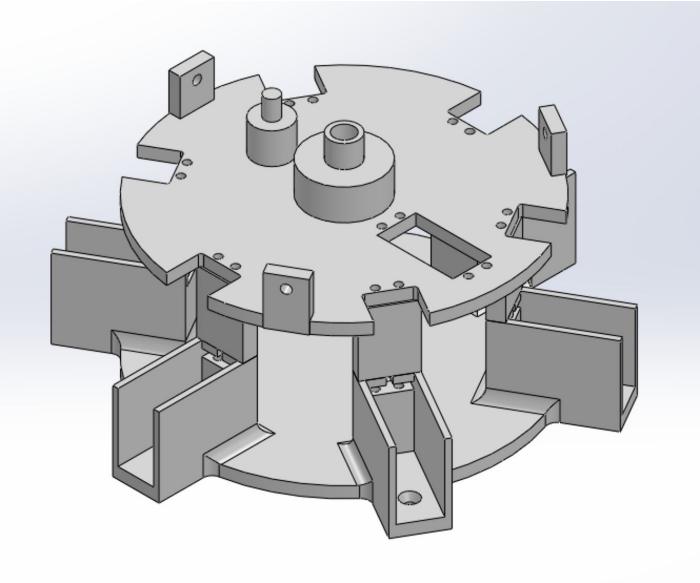


Figure 2.2.1: *Full base subassembly of the Guardian robot*

As shown by Figure 2.2.1 above, the base subassembly for the Guardian Robot is made up of three distinct types of parts. These three parts are: the bottom plate, the top plate, and the spacers connecting the top and bottom plates. Note that the base features of both the top and bottom plates of this subassembly are circular plates, and because of this, a circular geometry is imparted to the base subassembly as a whole. It was desired to have the base subassembly be circular in nature and not rectangular, not only to remain true to the original source material but also to help facilitate the desired omni-directional movement capabilities of the final robot. With a circular base, all six of the robot's legs could be distributed evenly across the circumference of the base, and thus, there would be no true "front" or "rear" ends to the robot. Because of this geometry, the robot would not have to rotate in place in order to change its direction of travel and could instead navigate direction changes with minimal interruption.

Figure 2.2.2 depicts an isolated view of the bottom plate of this subassembly.

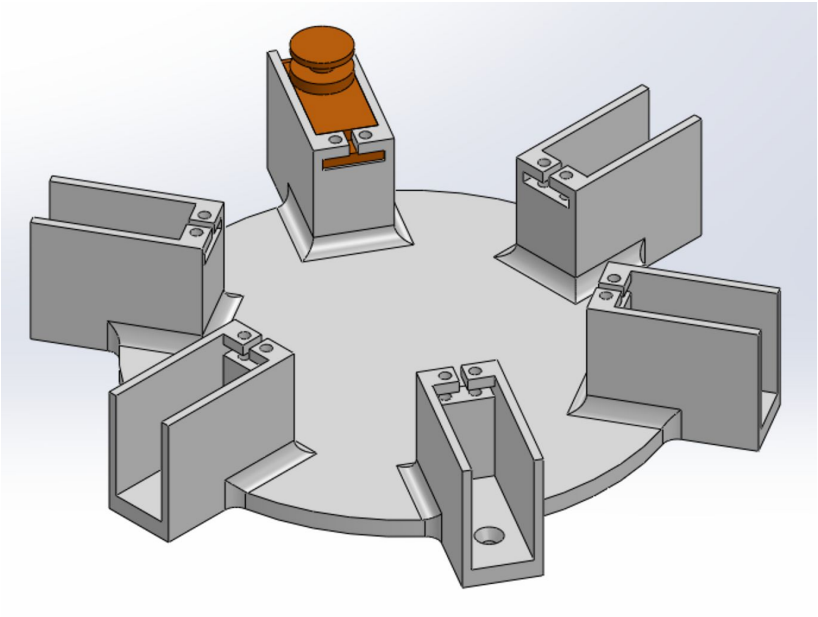


Figure 2.2.2: *Bottom plate of the base subassembly*

As shown by Figure 2.2.2, the bottom plate of the base subassembly consists of a circular base plate with six axisymmetrically-distributed, rectangular spokes protruding from it. The circular base is 7 inches in diameter and is 0.25 inches thick. The six spokes on the base plate serve as the mounts that hold the "hip joint" servos (orange part) in place and connect the base subassembly to each of the six leg subassemblies. Each of these spokes consists of a large slot in which the main body of the servo can sit. The back wall of each slot contains a secondary slot that holds the rear tag of each servo. Two 4 mm holes are drilled through each of these secondary slots to allow M4 bolts to be installed to hold the servos in place. Note that in the final assembly of the robot, the M4 bolts used were 50 mm in length and went through the top plate, the spacers, and the bottom plate, and thus, in addition to holding the hip joint servos in place, these bolts also held the entire subassembly together. Finally, chamfered holes at the bottom of each of the six spokes on the bottom plate allow flat head 10-32 bolts to be used to hold the first link in each of the six leg subassemblies to the bottom plate.

The figure below depicts an isolated view of the spacers used in the base subassembly.

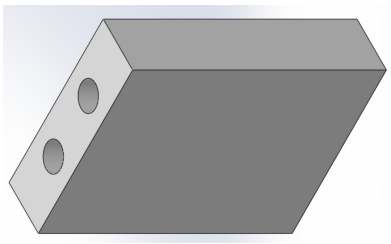


Figure 2.2.3: *Spacers used in the base subassembly*

As shown by Figure 2.2.3, the spacers used in the base subassembly are simple rectangular prisms with two through holes. The sole purpose of the spacers is to provide sufficient offset distance between the bottom plate and top plate of the base subassembly in order to allow for enough room for the interior of the base subassembly to house all of the electronics, batteries, and wiring necessary for the robot to function. In the final design, these spacers provided an extra 35 mm of offset between the bottom and top plates. Finally, the two through holes in each spacer are both 4 mm in diameter in order to allow the 50 mm long M4 bolts to hold the entire subassembly together.

Figure 2.2.4 depicts an isolated view of the top plate of the base subassembly.

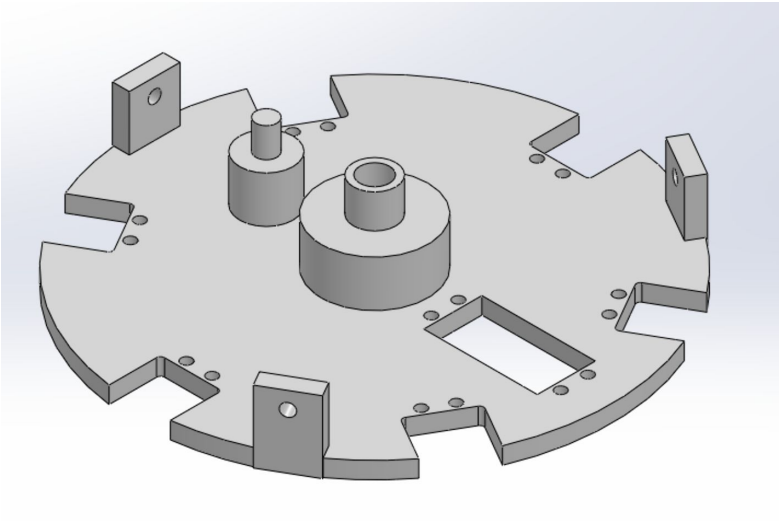


Figure 2.2.4: *Top plate of the base subassembly*

Like the bottom plate of the base subassembly, the base feature of the top plate of the subassembly is a cylinder that is 7 inches in diameter and 0.25 inches thick. However, instead of the six servo-mount spokes that were present on the bottom plate, the top plate instead features six axisymmetrically-distributed slots that simply exist to prevent the top plate from interfering with the legs' ranges of motion. Behind each of these six slots are two holes that are 4 mm in diameter. These holes exist to allow the 50 mm long M4 bolts to extend through the top plate and then down through the spacers and the bottom plate in order to hold the entire subassembly together. Thus, overall twelve M4 x 50 mm bolts were needed to hold the entire subassembly together. The top plate of the subassembly also features three axisymmetrically-distributed rectangular prisms extending upwards from the top of the plate. These three rectangular prisms exist to allow the outer turret shell (discussed in Section 2.2) to mount to the top plate via 10-32 bolts. Finally, the interior of the top plate contains two additional extruded features and one additional cutout. The cutout feature exists to allow the continuous servo responsible for the turret's 360-degree rotational motion to bolt into the top plate via four M4 bolts. Attached to the

horn of this continuous servo is a spur gear (discussed in Section 2.2) that drives a gear train to create the turret's rotational motion. The two extruded features on the top plate are the locations at which the other gears of this gear train are mounted. Finally, the hole through the central extruded feature on the top plate exists to allow an electrical slip ring to connect the electronics in the rotating turret to the batteries in the non-rotating base assembly.

2.3 Leg Subassembly

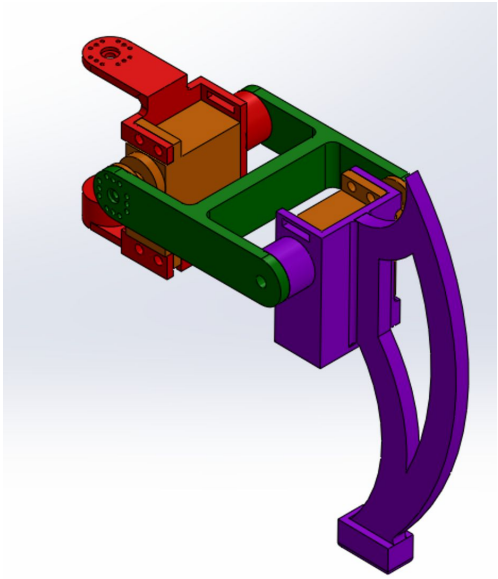


Figure 2.3.1: *Full leg subassembly of the Guardian Robot*

Figure 2.3.5 shows that, excluding the servo motors used to actuate the leg, the leg subassembly of the Guardian Robot is made up of three different segments. Segment 1 shown in red is connected to the bottom plate of the base assembly via a hip joint servo motor (not shown) that allows Segment 1 to rotate side to side in plane with the base assembly. Segment 1 is also connected to one end of Segment 2 (green) via a second hip joint servo motor that allows Segment 2 to rotate up and down in a direction perpendicular to the plane of the base assembly. Finally, the second end of Segment 2 is connected to the foot (purple) via a third knee joint servo motor that allows the foot to rotate on its own relative to Segment 2. With this setup, the motion of the leg assembly as a whole is able to achieve three degrees of freedom.

Figure 2.3.6 depicts an isolated view of Segment 1 of the leg subassembly.

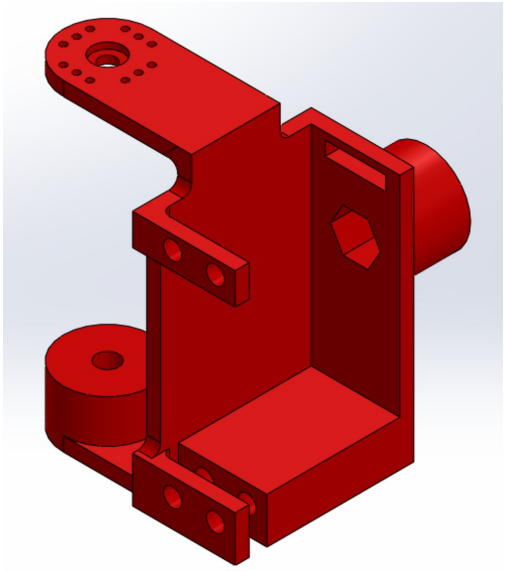


Figure 2.3.2: *Segment 1 of the leg subassembly*

As shown in Figure 2.3.6, Segment 1 of the leg assembly consists of a hollow rectangular section with tabs protruding out of the top and bottom of this section. These protruding tabs are what connect Segment 1 to the base assembly. The top tab connects to the horn of the servo mounted in the bottom plate of the base assembly via two M2 screws and one M3 bolt, and the bottom tab connects to the bottom plate via the 10-32 flat head bolt. Finally, the large, hollow rectangular section holds the servo connecting Segment 1 to Segment 2 via four M4 bolts, and the cylindrical extrusion on the outside of the rectangular section serves to hold a 10-32 button head bolt that creates a pin joint between Segment 1 and Segment 2.

Figure 2.3.7 depicts an isolated view of Segment 2 of the leg subassembly.

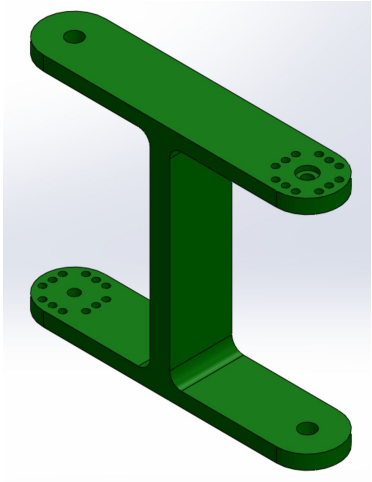


Figure 2.3.3: *Segment 2 of the leg subassembly*

As shown in Figure 2.3.7, Segment 2 of the leg subassembly is a symmetrical part consisting of a single U-shaped bracket that has been mirrored and flipped across a central plane in order to create two conjoined U-shaped brackets that ultimately form an “I” shape. Each U-shaped bracket has two ends. At one of the ends is a single, central through hole, and at the opposite end is a central counterbored hole with twelve smaller through holes surrounding it. Note that each of the U-shaped brackets that make up Segment 2 connect to either Segment 1 or the foot of the leg subassembly. The end of the U-shaped bracket with the counterbored hole and the twelve surrounding through holes connects to the horn of the servo that is mounted in either Segment 1 or the foot via two M2 screws and one M3 bolt. Meanwhile, the end of the U-shaped bracket with the single through hole connects to the cylindrical extrusions on the outsides of either Segment 1 or the foot via the pin joint created by the 10-32 button head bolt.

The figure below depicts an isolated view of the foot of the leg subassembly.

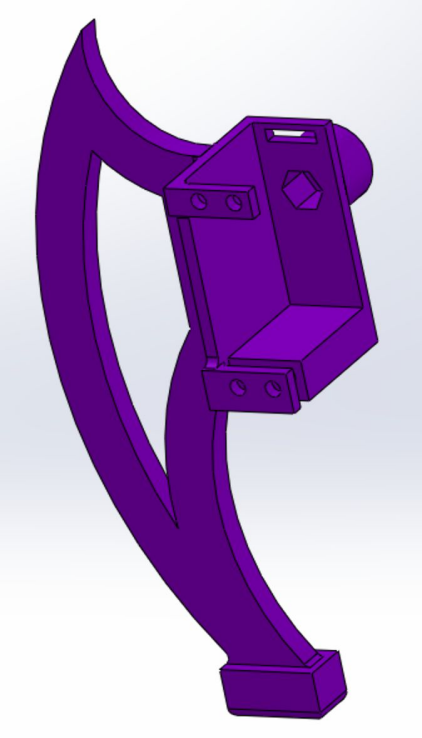


Figure 2.3.4: *Foot of the leg subassembly*

As shown in Figure 2.3.8, the foot of the leg subassembly is extremely similar to Segment 1 of the subassembly as both parts contain a hollow rectangular section designed to hold a servo motor via four M4 bolts. Moreover, both the foot and Segment 1 also contain cylindrical extrusions on the outsides of these rectangular sections to house the 10-32 button head bolts that create the pin joints between the segment in question and Segment 2. However, unlike Segment 1, the Foot of the subassembly does not possess the two tabs protruding from the back of the rectangular section that allowed Segment 1 to connect to the bottom plate of the base

assembly. In place of these tabs, the foot simply has a solid structure that contacts the ground during the robot's motion.

2.4 Turret and Imaging System

In order to accurately match the design of a Guardian from *The Legend of Zelda: Breath of the Wild*, a simple turret system was designed. A continuous servo bolted into the top plate of the main assembly drives a spur gear that is coupled to the teeth of a printed turret base plate. To make the access into the circuitry of the robot easier, the turret assembly itself was designed so that the top half of the turret can be easily dismounted from the rest of the body. This was accomplished by organizing most of the wiring and the main microcontroller inside the main body, with only several data transmitting wires connected to the sensors and servos in the turret via an electrical slip ring.

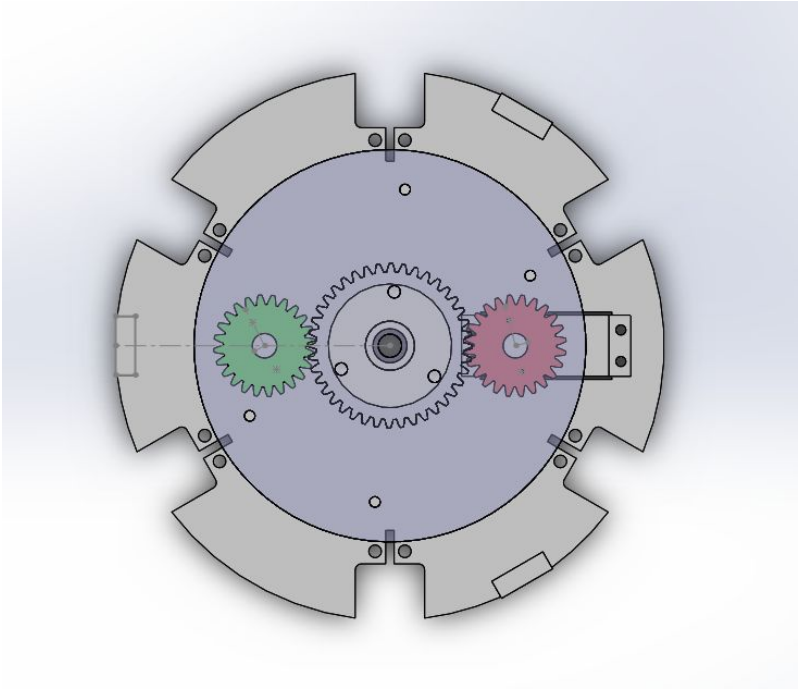


Figure 2.4.1: *Top down view of the rotating turret base plate*

Mounted on top of the rotating base plate of the turret assembly was an upper table upon which the pan and tilt mechanism (described in a later section) was attached. This upper table also allowed for the mounting of an outer shell that served to protect the pan and tilt mechanism as well as mount the camera to the front of the robot. The upper table and the outer shell are both shown in Figure 2.4.2 on the following page.

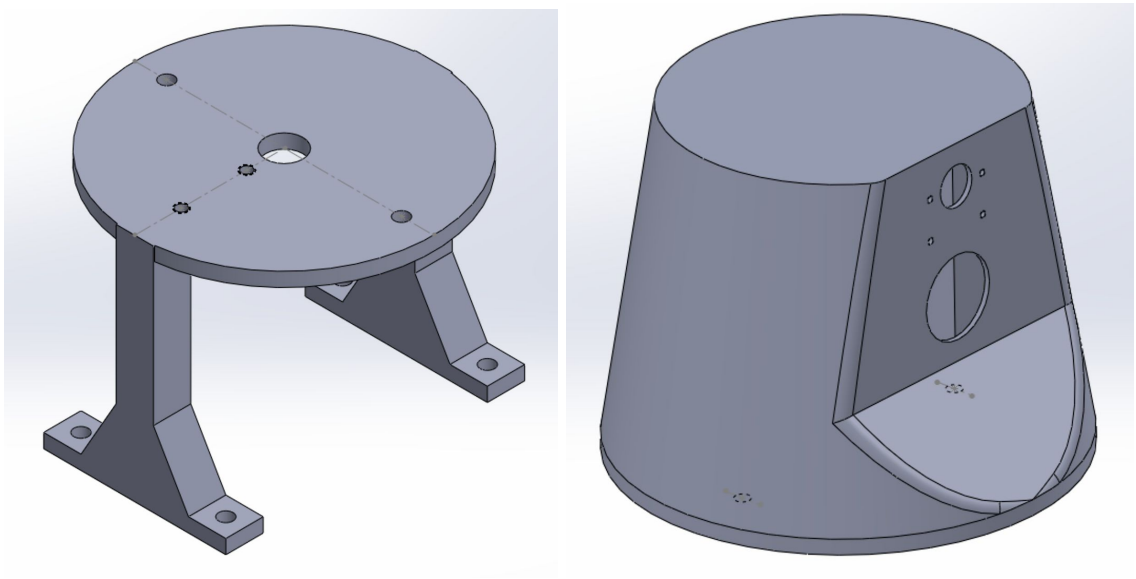


Figure 2.4.2: *Upper table and outer shell of the turret subassembly*

When fully assembled, the entire turret subassembly consists of the continuous servo driving a spur gear, the turret base plate mated to the teeth of this spur gear, the upper table mounted on top of the rotating base plate, the pan and tilt mechanism mounted on top of the upper table, the outer shell protecting the pan and tilt mechanism, and the camera mounted to the exterior of the turret. An image of the full turret subassembly is provided in Figure 2.4.3 below.

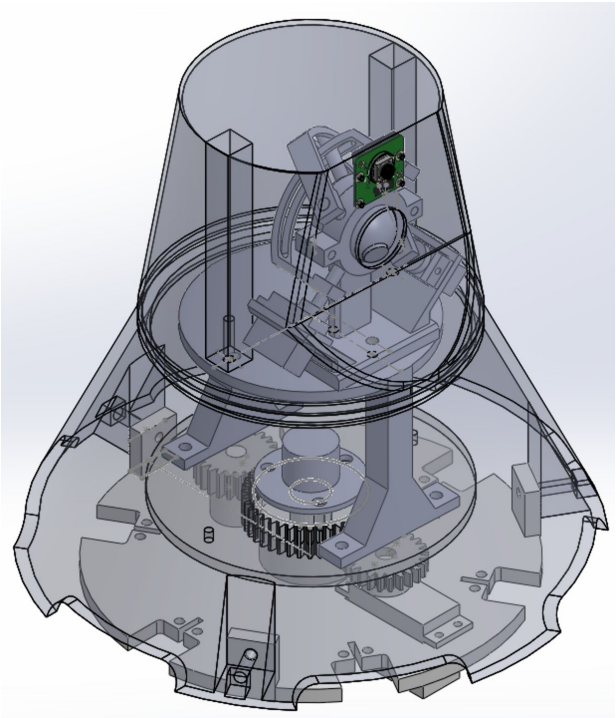


Figure 2.4.3: *Full turret subassembly*

The following subsections discuss the electrical and mechanical specifications and usage of both the external camera and the pan and tilt mechanism of the turret subassembly.

Camera

The camera is a standard Raspberry Pi camera that constantly takes pictures (frame of 960 by 720 pixels - smaller frame for faster transmission through sockets) and sends it over TCP/IP to the computer for image recognition and detection. The camera feed can be seen in real-time on the web dashboard on the computer.

Laser Pointer + Pan and Tilt

The pan and tilt mechanism of the turret subassembly was used to hold the Guardian's laser pointer and allow it to move with two rotational degrees of freedom. This mechanism was also designed so that these two degrees of freedom (the pan and the tilt) would be completely separate from each other. An image of the full pan and tilt subassembly is shown in Figure 2.4.4 below.

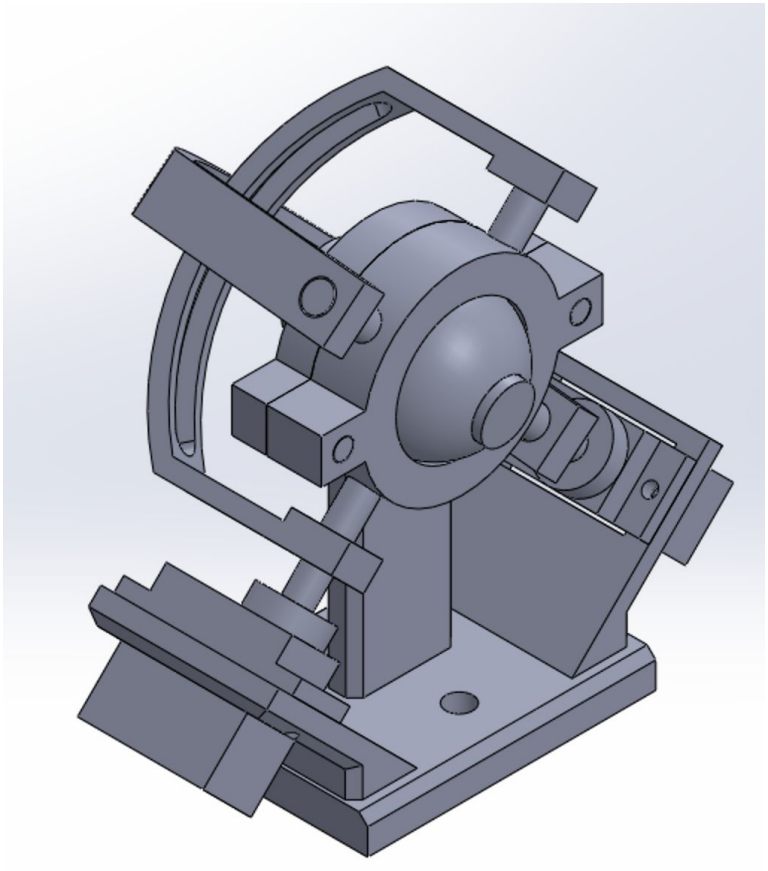


Figure 2.4.4: *Full pan and tilt assembly*

As shown by the figure on the previous page, the pan and tilt assembly consisted of a single sphere with a cylindrical shaft extending out of its back. This sphere rotated inside a chamfered circular bearing, allowing it to achieve the desired rotational degrees of freedom. To actuate the rotational motion, the two "C-rail" parts were used. These C-rail parts each had a curved slot cut into them to allow the cylindrical shaft of the main sphere to fit through them. By doing this, the two C-rail parts could then be actuated with two micro servos, and the rotational motion of the C-rails could then be converted to rotational motion of the shaft, the sphere, and ultimately the laser pointer.

In our original implementation for this component of the robot, the laser pointing mechanism was created using a simplified PD control system. The camera would take pictures with a resolution of 960 by 720 pixels, where the object detection program would derive the pixel coordinates of the central position of the Link figurine. The coordinates of where the laser is pointing would also be found using this method, giving a vector that the laser must move to make the Link coordinates and the laser coordinates line up. This vector would then be passed through a rotation matrix that tilts the coordinate axis by 45 degrees to accommodate for the angles at which the pan and tilt servos were mounted onto the turret. After the transformation, the individual components of this vector would be taken and applied to their respective servos to adjust the amount of error that is currently present in that axis. This process is summarized in the following figure:

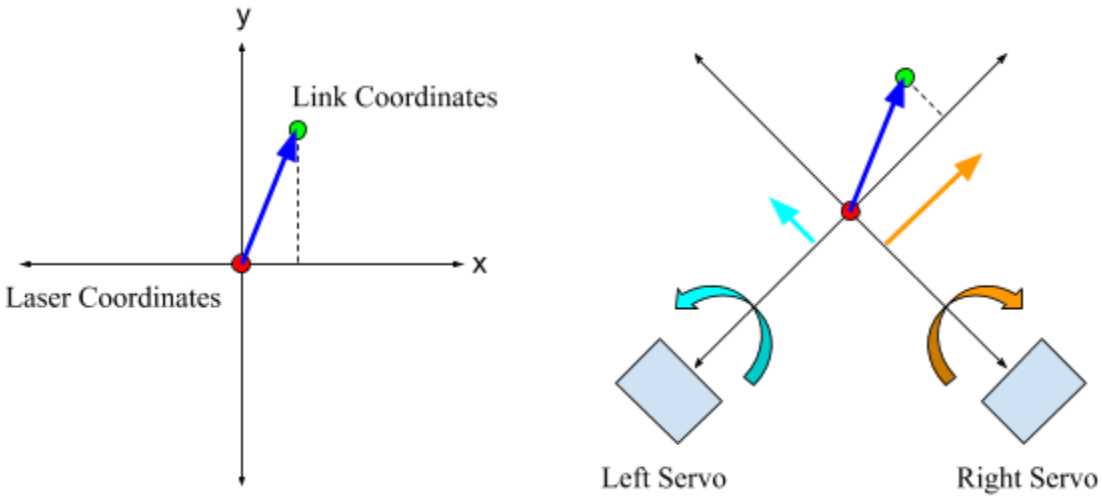


Figure 2.4.5: Laser Pointing Mechanism *- The left figure represents the pixel coordinate system directly from the picture taken, with the blue arrow indicating the vector in which the laser must move in order to align with Link's coordinates. The right figure represents the new coordinate system after the rotation, allowing for the light blue and orange vectors to be derived to move the respective servos in the correct direction.*

With the original implementation, we ran into some complications with getting feedback on the laser position and with the deadline quickly approaching, we went for a different, simpler solution. We scaled the max ranges of both servos to the 960 by 720 pixel frame. At startup time, the laser points at the center of the frame. Once we got Link's location (in terms of pixels), we used ratios to determine the servo positions, ultimately getting the laser to point at Link. Every time Link's coordinates change, the servos move accordingly, so the laser can continuously point at Link.

2.5 Object Detection and Classification with YOLO

The You-Only-Look-Once algorithm is motivated by the need for fast detection speed. It was chosen because it massively outperforms the rival algorithm SSD (Single Shot Detection) in terms of raw speed. The YOLOv3 real-time object detection architecture and Darknet open source neural network framework were originally created by Joseph Redmon and Ali Farhadi from University of Washington. A fork of the project by AlexeyAB on GitHub was utilized for the Python wrapper of the framework. For even faster detection, a simplified variant of the architecture, YOLOv3-tiny was used for the Link Amiibo detection model. The model was trained on 54 labelled images of Link over 2000 epochs. Through testing, model accuracy is superb, however the detection latency can be up to a full second when running on a laptop without a dedicated GPU.

As shown by the following detection results, the model is able to detect Link at varying angles, distances, and backgrounds. There were initial concerns about overfitting because of the limited training set, but after testing with models cut off at 100, 1000, and 2000 epochs, the final model still performed the best during testing.

Figure 2.5.1: *YOLOv3-tiny Detection Results on 4 Training Images*

2.6 Object Detection and Classification with TensorFlow

Tensorflow provides a library to allow for symbolic math and programming for a variety of purposes, especially for machine learning. The library is very powerful and allows custom models to predict and compute. Using pre-trained models for image classification from the following image datasets: the COCO dataset, the Kitti dataset, the Open Images dataset, the AVA v2.1 dataset and the iNaturalist Species Detection Dataset, the detection model is very powerful in classifying Link but also many everyday objects.

Using two different models of different performance and speed, we were able to finalize on a model to do the image classification. Using the model “faster_rcnn_inception_v2_coco” for better performance and the model “ssd_mobilenet_v1_ppn_coco” for better speed, we had two models to find Link given a video stream or images. The success rate from the more performant model was over 90% but it takes around a minute to process 50 images. The faster model had a success rate of over 80%, but it was 3x faster, as it only took 20 seconds to process the 50 images. Distance, image quality, and color made a big difference with the success rate, but overall the models were able to detect Link in a variety of conditions.

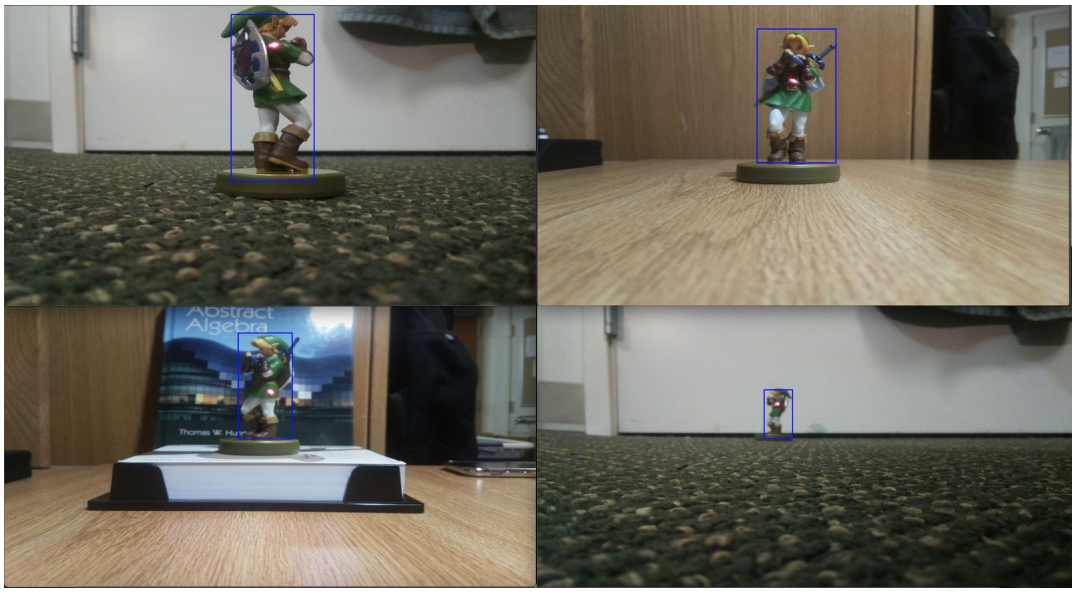


Figure 2.6.1: *Tensorflow Detection Results on 4 Training Images*

Currently, we have both YOLOv3 and Tensorflow available to use for object detection. Though YOLOv3 is known to be traditionally faster than Tensorflow, Tensorflow provided us with better accuracy. Considering that the camera is placed on the turret, we need the turret to move slow enough so that each frame the camera captures will not be blurry. Due to this, we value accuracy more than faster processing, and ultimately decided on using Tensorflow.

2.7 Sockets and Data Flow

This project only requires three devices: two Raspberry Pis and a computer. All three are connected to a local router, and with Local Area Network, these devices transfer data among each other via TCP. The Raspberry Pi Zero sends image data to the Python backend on the computer. This image data is then sent over to the web dashboard's nodeJS backend. The frontend of the web dashboard communicates with the nodeJS backend via Websocket, to display the images on the dashboard. Additionally, the controls that the user inputs on the dashboard is sent over Websocket from the frontend to backend of the web dashboard, and then over to the Python backend. The Python backend takes the image data (from the Raspberry Pi Zero) and the input controls (from the web dashboard), processes this data, and sends the necessary position and motion data to the Raspberry Pi 3. This flow of data can be seen in Figure 2.7.1.

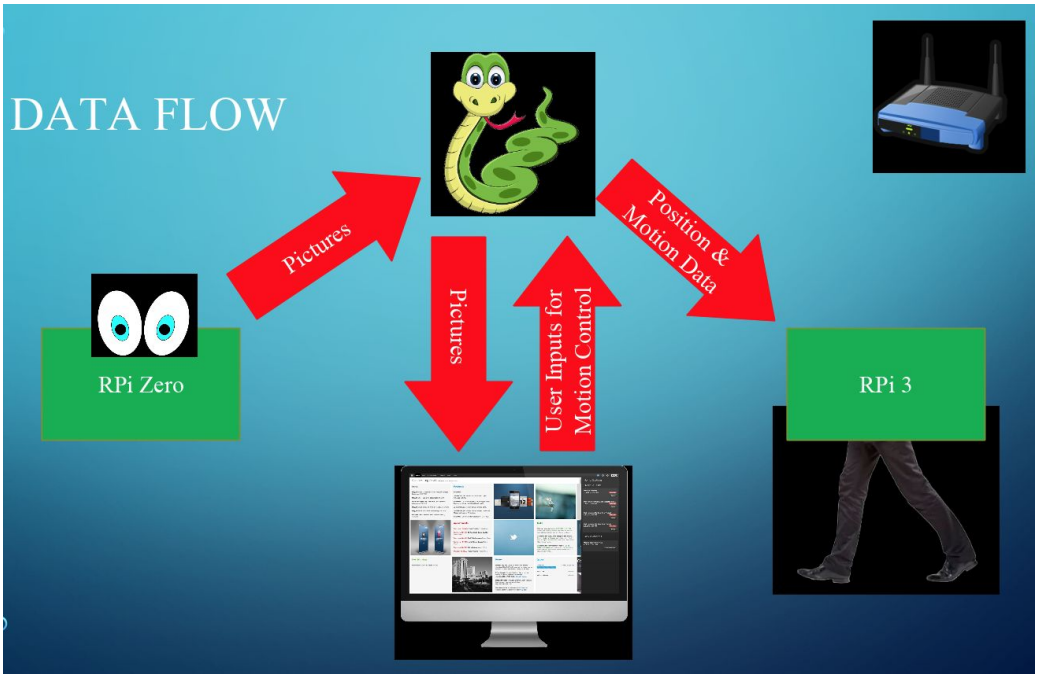


Figure 2.7.1: *Flow of data between the computer and Raspberry Pi's*

2.8 Web Dashboard and Commands

The web dashboard is a React app with authentication via Firebase and design using Materialize CSS. Upon signing in, a user can access the dashboard component, which features a livestream from the Guardian on the left and a motion control input interface on the right. On the right, users can toggle on and off turning of the head, input body turn degree and direction, input the distance they wish the Guardian to travel, and input the movement angle degree and direction. There are also displays of updated laser position and link position at the bottom left corner at each moment.

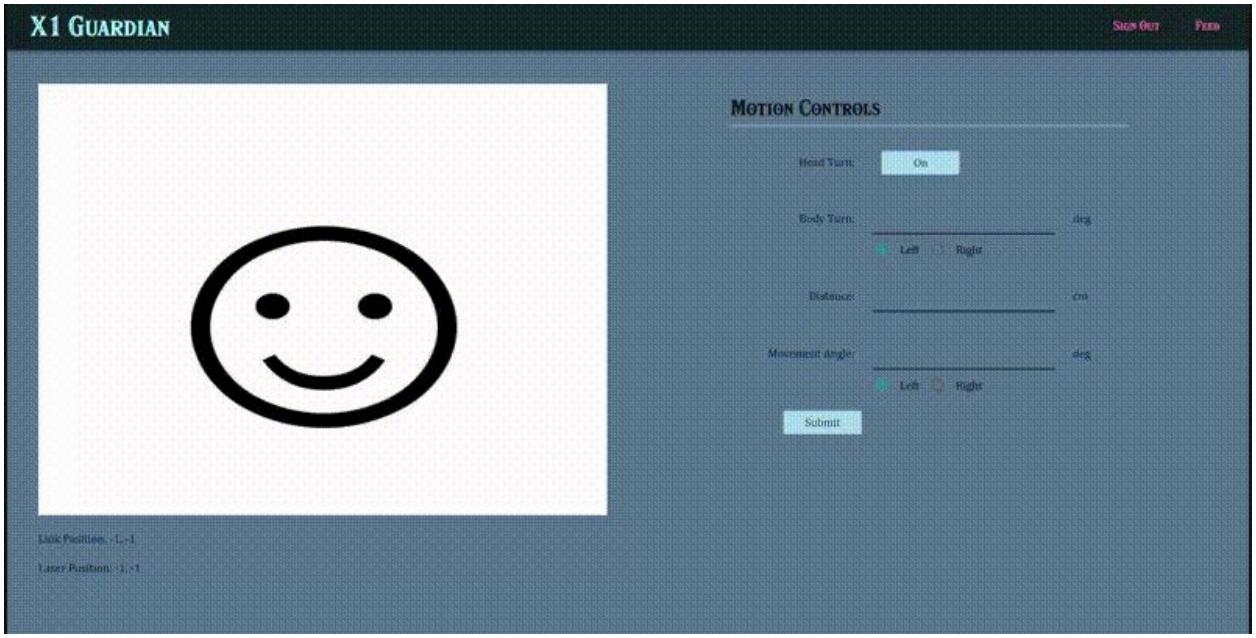


Figure 2.8.1: *Running a test on web dashboard to display different images back and forth*

From the web dashboard, two types of commands are sent to Python when the user provides motion control input. If the user toggles the head turn button, a string is sent indicating whether it is being toggled ON or OFF. If the user hits submit, a comma-delimited string is sent with fields in the order "<Body Direction>, <Body Degree>, <Distance>, <Movement Angle Direction>, <Movement Angle Degree>". Directions are indicated with "L" for left and "R" for right; degree and distance are indicated by their integer values.

2.9 Controls

The legs of the Guardian robot consists of three servos, which gives it three degrees of freedom. The base servo allows the leg to rotate in the xy plane, while the other two give it vertical freedom that allows the legs to lift off of the ground.

The Guardian robot utilizes an open-loop leg control system that does not receive any feedback about whether or not the leg was successfully moved to the desired position. Thus, the implementation of the walking animation merely consisted of being able to calculate the correct servo angles that follow the specified 2D walking animation that is hard-coded into the program prior to running. Each leg has its own animation plane that's rotated in the z-axis according to the direction that the user wanted to walk in, allowing for the same walking animation to be reused to allow the robot to walk in any desired direction, regardless of the orientation of the robot relative to the world. This idea is shown in the following figure:

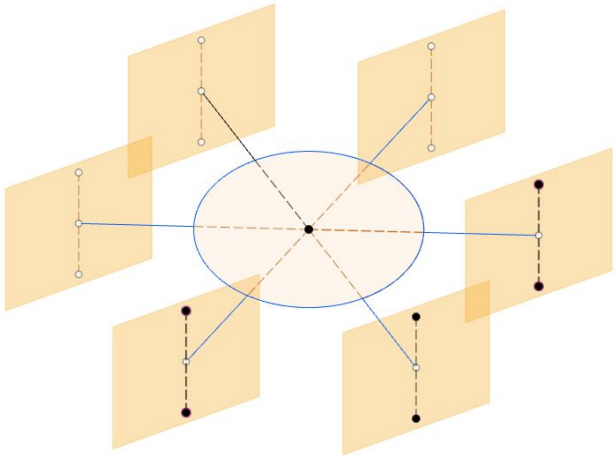


Figure 2.9.1: *Illustration of Animation Planes Relative to Body of the Robot – The circle in the middle represents the main body of the robot that holds the electronic components such as the Raspberry Pi, the batteries, etc. The orange squares represent the animation planes that the animations were drawn onto and followed by the legs using inverse kinematics. These planes were rotates along the z-axis (vertical dotted lines) so that it is possible to change the direction of travel on the spot without any need of turning the body itself.*

These 2D walking animations were stored within a text file which consisted of multiple coordinates that showed the position of the leg for each animation frame. A dedicated application for creating the walking animation was created using a software called Processing, though we ended up sticking to a simpler solution of making the animation into a simple half-circle shape due to convenience. The program outputted a text file that consisted of coordinates that were centered at the average point across all coordinates, and had a maximum magnitude of 1.0 so that the main program could scale the animation at will. A sample image of the editing screen of the application is shown in Figure 2.9.2.

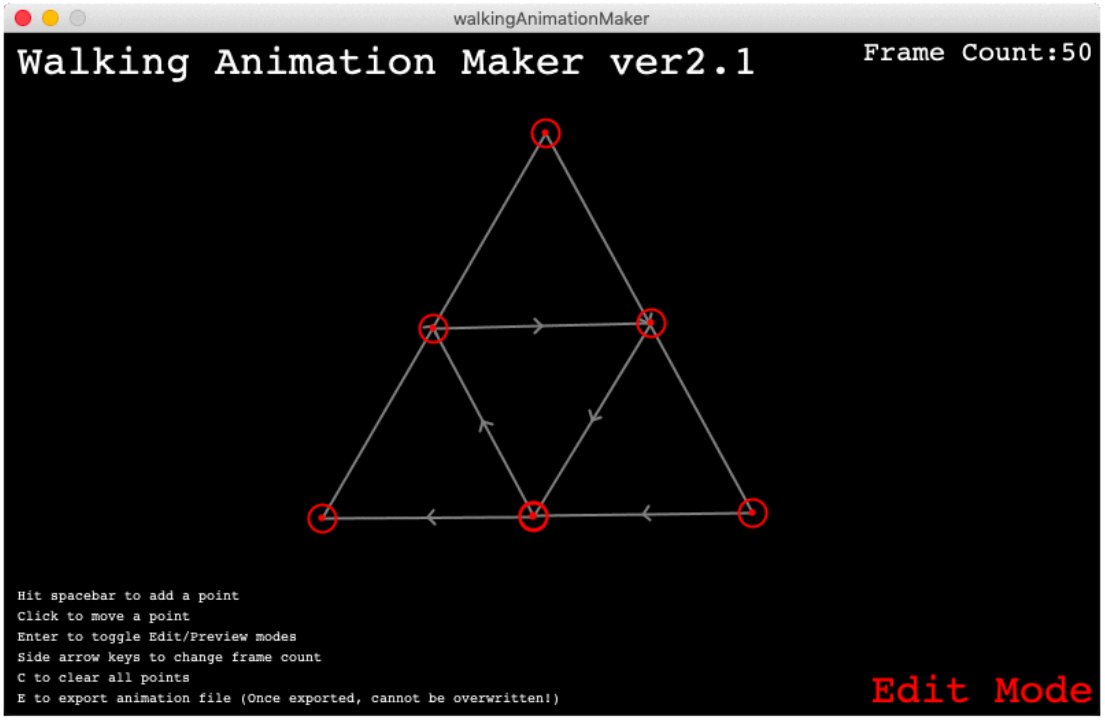


Figure 2.9.2: *Walking Animation Maker Software – Although we did not end up using it for the final product, this program was useful in the initial stages of the project where we were testing out the motions of a single leg by drawing out the trajectory we wanted the leg to follow and verifying it using both a model and the actual leg.*

As described above, inverse kinematics was used to calculate the servo angles of each of the legs. Each leg contains three segments, which are actuated by three servo motors. The problem then can be modeled as finding the three angles of the servos such that the tip of the leg reaches a point specified in 3D space, given some constraints. If we consider the 3D space using a cylindrical coordinate system, we see that the angle of the base servo is the azimuthal angle of the specified point. Then the problem is simplified to finding two angles such that tip of the leg reaches the specified point in a 2D space, given some constraints.

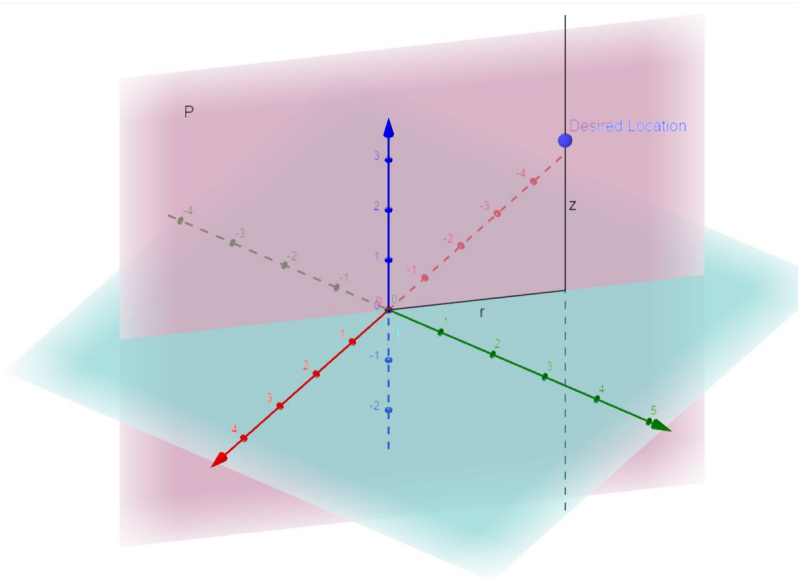


Figure 2.9.3: *Animation Model in 3D – Because the range of motion of the base servo is limited to the XY-plane, we can specify a plane P for every desired location given in 3D coordinates. The azimuthal angle of the plane will be the actuation angle of the base servo. The segment motions from actuating servo 2 and servo 3 will be entirely in the plane P.*

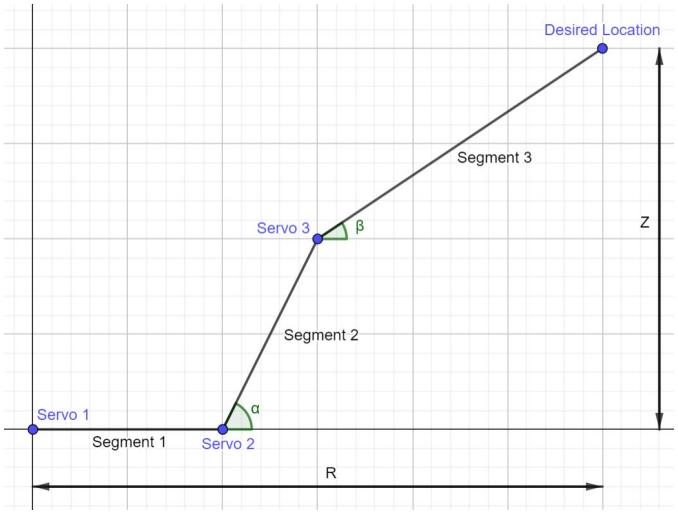


Figure 2.9.4: *Animation Model in 2D – The problem reduces to 2D when we consider only what happens in the plane P. The parameters R and Z are computed as in Figure 2.5.3, and length of the segments are the physical constraints of the system.*

From this, we can write a system of two nonlinear equations relating the angles of servo 2 and servo 3 to the position of the tip of the leg, given the constraints of the segment lengths:

$$R = s_1 + s_2 cos(\alpha) + s_3 cos(\beta)$$

$$Z = s_2 sin(\alpha) + s_3 sin(\beta)$$

To analytically solve the system, we need to make use of a computer algebra system (CAS), which is not feasible to do in real time on our hardware. We decided to use Newton's Method to solve the system, which is a numerical algorithm using non-linear least squares optimization to iteratively approximate the solution. Given that there are multiple solutions to the system, further constraints are added to force the algorithm to yield the desired solution. Hyperparameters such as the number of iterations can be used to further control the trade-off between algorithm speed and accuracy. The final implementation use on the robot is fast enough such that the angle computations for all 18 servos (6 legs × 3 servos per leg) can be done in real time without noticeable lag time in robot movement. The accuracy achieved by the algorithm is also much higher than the actuation resolution of the servos, which means the error between the exact solution and our approximate solution will not lead to any difference in servo actuation, which is the desired result.

In order to make sure that the servo angles are within a reasonable range that will not destroy the leg itself, a Python simulation was written to view the leg positions prior to uploading the program to the actual robot. This simulation was also used to test out the tripod walking gait before we even got the physical leg to test the program on. The simulation is shown below:

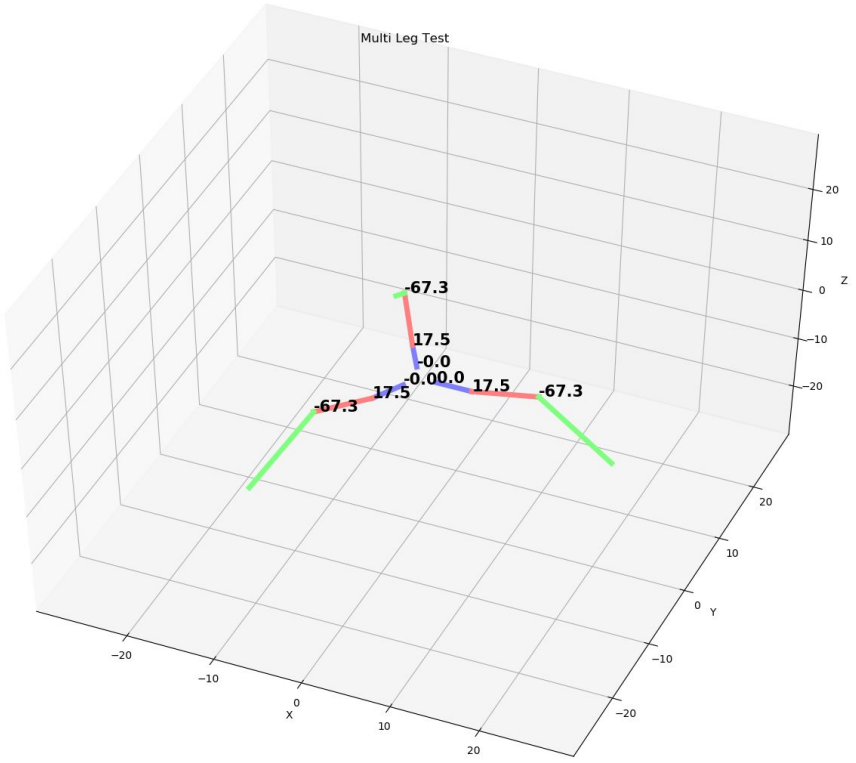


Figure 2.9.5: *Python Simulation of Guardian Robot (3 Leg Version) – Each of the different colors represents a different segment of the leg, and the numbers next to each joint describes the*

angle of the servo at this moment. Currently running a program that assumes only 3 legs to be used, but this can be freely changed via parameters passed through when launching the program through the command line.

2.10 Electrical Design

Due to the need to control a large number of servos at once, we decided to use a Raspberry Pi 3 with two 16 channel servo hats as the main microcontroller for this project. We also opted for this design because we had anticipated that the image recognition would be taking place on the controller, though we ended up using two separate Pi's for the leg control and the head. Since the servo hats required a power source on its own, we used two separate batteries for the hat and the Pi 3 so that the possible noise from the servo hat could be isolated from the power source of the microcontroller. A switching voltage regulator from Adafruit was used to step down both voltages to 5V with a maximum draw current of 3A, which after experimentation was determined to be enough to sustain 9 servos on a single hat. Thus, two voltage regulators were connected in parallel for the servo batteries to be able to move all 18 servos + 3 servos used to move the head and the turret mechanism.

Being true to the original robot from the game, we wanted the robot to have full 360 degrees freedom in the head, thus being able to spin the head limitlessly using a continuous servo. For this reason, a Raspberry pi zero was used to take the images and send them to the web dashboard, along with using a slip ring that provided signal and power to the two micro servos that controlled the position of the turret without having the wires tangle up. This made it so that the head and the body were completely detachable, helping us when developing each, different part of the robot.

3 Conclusions

Overall, the 2018-2019 build cycle for X1 Robotics' Guardian was a success. The team accomplished all of the tier one and tier two design objects discussed earlier. The robot was capable of navigating three-dimensional space, supporting its own weight, and rotating. It was also capable of locating and classifying its Link amiibo target. Lastly, the web dashboard GUI was developed, allowing commands to be remotely sent to the Guardian.

The third tier of design objectives were not met due to time constraints. Though shipping times were factored into the project's timeline, unpredictable complications delayed certain phases of testing and assembly. These issues will be factored into time allocation for the upcoming year. Furthermore, X1 Robotics will continue working on the Guardian robotics system in the 2019-2020 build cycle to complete all the design objectives set forth at the start of

the 2018 academic year, simultaneously undertaking a new system to uphold the club's value of creativity.

Over the course of the build cycle, the safety and well-being of the team members was ensured. Timelines were planned with flexibility in mind as to not hinder the academics of any X1 participants. Additionally, standard protocols were followed as outlined by the UCLA Samueli School of Engineering and UCLA Environment, Health, and Safety (EH&S).