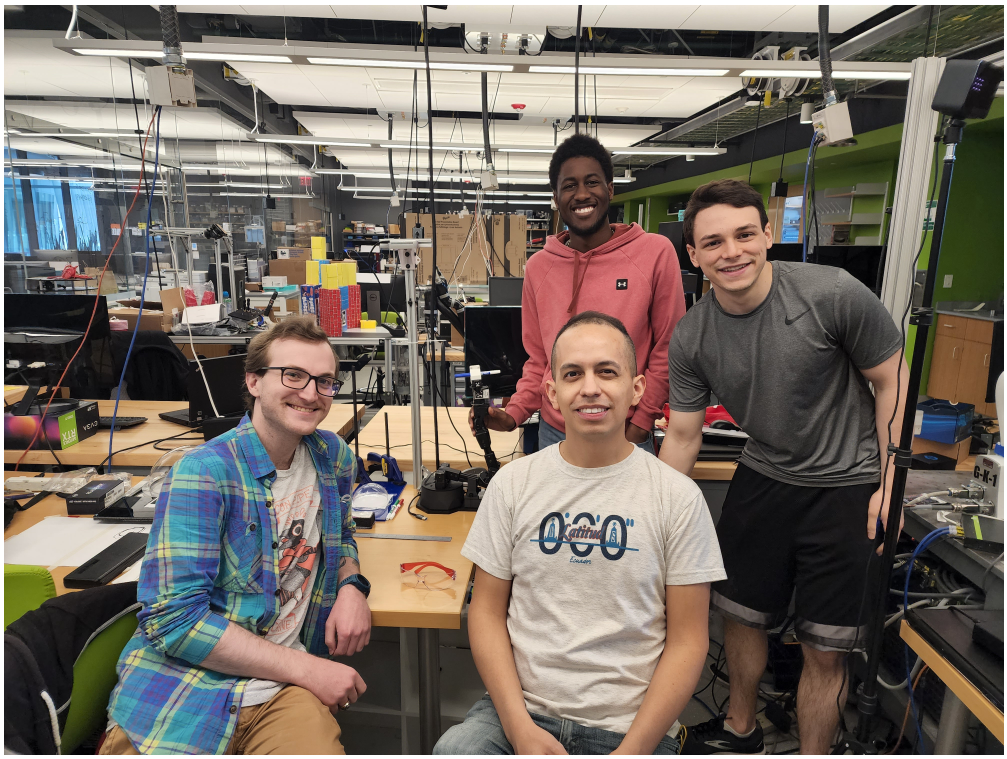
CS-5335 MichelARMgelo Final Report

James Tukpah, Kevin Robb, Michael Carvajal, Richard Kaufman

Spring 2022

This report details our use of an Interbotix PincherX 150 robot arm to draw with a marker on a variety of surfaces. All code we wrote for this project, as well as a full technical setup guide, is included in our GitHub repository. A short video montage that includes several demonstrations of our progress and final product is available on YouTube. This was our final project for CS-5335: Robotic Science & Systems at Northeastern University.



Contents

1 Introduction

1.a Project Outline

It is common in this day and age to utilize robotics in areas where automation is desired. Fields such as manufacturing, packaging and shipping, and testing are seeking massive developments in robotic arm automation specifically. The capability of a robotic arm to manipulate and interact with its surroundings is something to be amazed at, and the usefulness of such features far extend outside of simple automation. There is currently research for the use of robotic arms in surgery because of their ability to maintain a constant position and a superior level of precision. There is also an effort to utilize robotic arms in construction due to their ability to swap out various end effectors. What our team was interested in is the possibility of taking all of these functions and applying them in a more unique and under-researched application for robotics: the arts!

The various uses of a robotic arm would fit perfectly with the array of needs that art requires from artists. This thought birthed the concept of a great robotic artist: MichelARMgelo. The purpose of this system would be to explore the possibilities of using a robotic arm for the development of art. The possibilities seemed endless to us. The precision of the arm would be perfect for etching, sketching and drawing various illustrations. All the while, the ability of the arm to perceive information through a camera would make it possible to replicate any writing, painting, or drawing that was shown to the system, if using a good enough algorithm. The prospects were abundant; however, our time, finances, access to appropriate equipment, and knowledge were not.

1.b Components and Constraints

Given a one-month timeline, the extent to which our group could develop an artistic robot was limited. While we were generously lent equipment, there were still limitations on its performance, such as its available workspace and maximum motor torques. With constraints in mind, our group discussed what robot-drawing related projects we would be able to pursue. To do so, we outlined our available components, created a list of assumptions, and agreed upon additional pieces that we would need to manufacture ourselves. The primary piece of equipment was an Interbotix PincherX 150 (px150), a 5-DOF robotic arm which works with the pre-developed Interbotix ROS workspace designed for ROS Noetic/Melodic. Here, we assumed that the arm's base would be fixed to a surface, which we accomplished by clamping it to the edge of a tabletop. We procured a dry erase marker as our tool, and a flat whiteboard and two plastic domes to serve as our canvases. For all drawing matters, we assume that the canvases are also secured to a surface, and will be reachable by the arm. For the domed canvases, we also assume their centers are in line with the arm's $Y = 0$ plane. For the robot arm to hold the marker as it draws, an additional component is required: an end-effector gripper of our design. Here we make an additional assumption that the marker's tip will always be at a constant known offset relative to the px150's default end-effector position. In addition to these components, we were provided with a RealSense D435 camera. To work effectively with the camera, we planned to print AprilTags and construct an overhead mount. When operating with the camera, we assume the robot arm and canvas will always be in full view, along with any AprilTags. What follows is a comprehensive list of the system components outlined herein:

1. Interbotix PincherX 150 arm and software packages
2. ROS Noetic or Melodic

3. Standard sharpie or thin expo dry erase marker
4. Custom 3D-printed end effector
5. Dry erase board and hemisphere
6. RealSense D435 camera and overhead mount
7. Printed AprilTags

After discussing the system components available and agreeing on the assumptions we would make, our group composed a goal for the project, in addition to several sub-goals, outlined in the next section.

1.c Project Goals

Our primary objective was to program the PincherX 150 arm to use a marker to draw on a canvas. To best fit this objective within the constraints outlined in section 1.b, and to track our progress, we collectively establish seven sub-goals for this project. From the constraints, we evaluate what may be feasible for our group to achieve. These goals function as a project milestones and get successively more complex. The intention is not necessarily to complete each goal, but instead to use them to gauge our progress and ensure we remain on course for our overall objective.

Project Goals:

1. Design a custom gripper to mount on the arm and hold the marker.
2. On flat paper, draw a variety of pre-planned shapes.
3. On the *exterior* of a domed surface, draw smooth arcs.
4. On the *interior* of a domed surface, draw smooth arcs.
5. Write alphanumeric characters.
6. Use AprilTag detection to localize the robot arm's end-effector pose.
7. Recognize and replicate a smooth path.

From this list we see that the goals sequentially increase in difficulty. We start with an easy goal that entails using the robot to draw simple shapes on a flat surface, and hope to end with drawing on non-flat surfaces, adding recognition, and drawing more complex characters. Our aim is to meet as many of the outlined sub-goals as possible, and to evaluate how we would plan to approach those that we did not reach.

2 Custom Gripper

2.a Functionality

The primary purpose of the gripper is to facilitate mounting the dry erase marker to the robot arm. The default pair of end-effector gripper fingers that come standard with the PX150 arm could not adequately hold the marker in place given that those fingers have flat mating surfaces but are being tasked with pinching the cylindrical exterior of the writing tool; cylindrical objects can only make tangential contact with flat

surfaces, and such a "lines-only contact" interface results in a minimal amount of tool stability while the pen is being gripped by flat-faced fingers. The weak grip between the default fingers and the writing tool is only exaggerated when the arm attempts to draw. As the tip of the marker exerts a normal force against the drawing canvas, the reaction force from the canvas acts on the gripper as a shear force pushing parallel to the flat faces of the gripper fingers. In order for the gripper to maintain the relative position and orientation of the marker caught between the fingers, the gripper's pinch motor would have to exert a substantial amount of torque such that the pinching force acting on the tool creates enough static friction to combat the shear reactionary writing forces. The PX150's pinch motor was simply not powerful to do so; to address this shortcoming, our team designed a custom pair of gripper fingers that maximizes the contact area between them and the writing tool.

2.b Design Requirements

The first set of design requirements involve mounting custom grippers to the robot arm's end-effector. Without any attachments, the end-effector of the robot consists of two carriages (P/N: SRG-Carriage-PXWXRX-PLA) for add-on grippers. These are mounted to a $10mm \times 10mm$ aluminum extrusion profile. The final stepper motor of the robot can move these two pieces symmetrically along the profile functioning as a revolute joint. Here there is a limited range of motion, as the two carriages cannot get too close or too far from one another. Both of these pieces contain two M2 holes intended to mount additional pieces for a gripper. The engineering drawing for these mounting points is displayed in Figure 1 below:

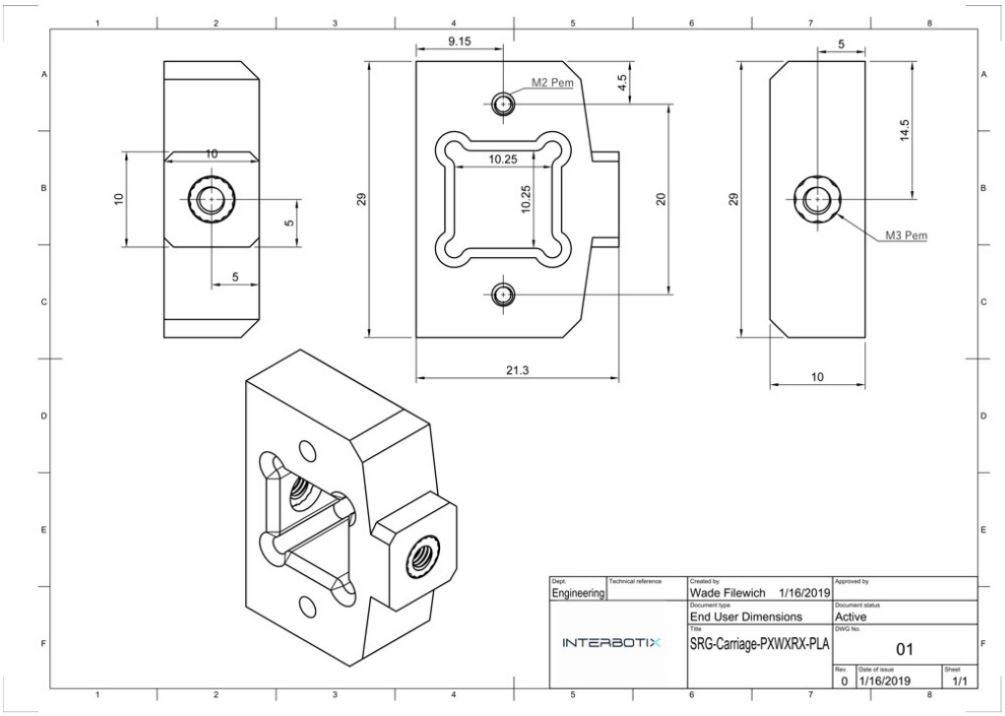


Figure 1: Carriage Engineering Drawing (from PincherX 150 Robot)

From these features, we set 3 design requirements.

1. The width of the gripper must be greater or equal to than half the minimum distance between the carriages

2. The width of the gripper must be less than half the maximum distance between the carriages.

3. The gripper's geometry must include two M2 holes to connect to its carriage and adequate space to fit into the aluminum profile.

In addition to mounting the gripper to the robot, it must be able to achieve its primary function: to hold the marker in place as it makes contact with the paper. To achieve this, the gripper's contact with the marker should produce a shear force greater than the component of the reaction force applied by the canvas on the marker that is in the direction of the marker's axis. The marker's surface must additionally have contact with multiple points along the marker's axis. This will produce a moment to counteract the moment induced by the components of reaction forces perpendicular to the marker's axis. In other words, it would prevent the marker itself from rolling and pitching as it writes.

2.c Gripper Design

To design the gripper, we first tackle the stability requirements. The piece is designed with V-groove cutouts on one end, added with the intention to hold the marker. This design guarantees two points of contact. We measure the diameter of the marker and create the V-slot cut such that the contact points are at a depth less than half its radius. A new problem presented itself here; the marker's diameter is nonuniform, and changes along its major axis. This may cause the gripper to only make contact with the marker at its cross-sectional circle with the largest diameter, which could make the pen to roll or pitch as it writes. To address this, we decide to increase the size of the V-slot and add 1.5 mm of foam to its inner surfaces. The foam will mold to the changing diameter of the mounted marker, alleviating the potential for the marker to rotate in the gripper. In Figure 2, the front view of the gripper is displayed in SolidWorks. The black portion of the part represents where the foam will be added.

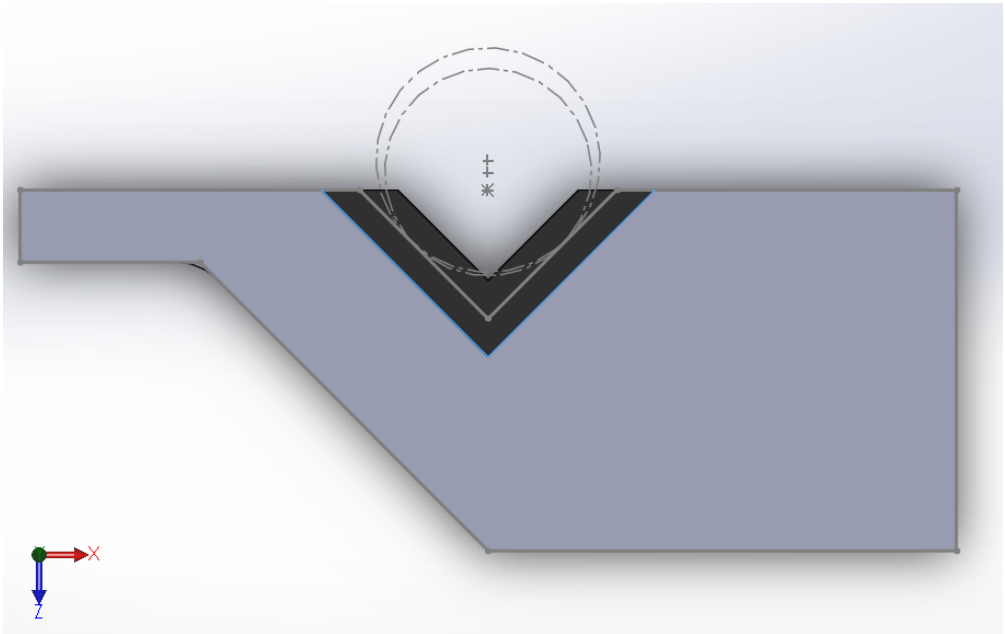


Figure 2: Front View of Gripper Model in SolidWorks

We next design the end of the gripper that will connect to the carriages. The PX150 arm documentation

includes drawings for the carriages, which we used to ensure the holes added to our gripper model will align. Additionally, a square slot was added to the end of the gripper model to ensure it fits around the aluminum extrusion profile.

Under the current model described, the movement of the carriages closer together, driven by the last servo motor, pinches the gripper. As an extra measure, two M3 holes were added to the opposite side of the gripper. These allow the grippers to be clamped together to supplement the force applied by the carriages. Below in Figure 3 we display the engineering drawing for the final gripper design.

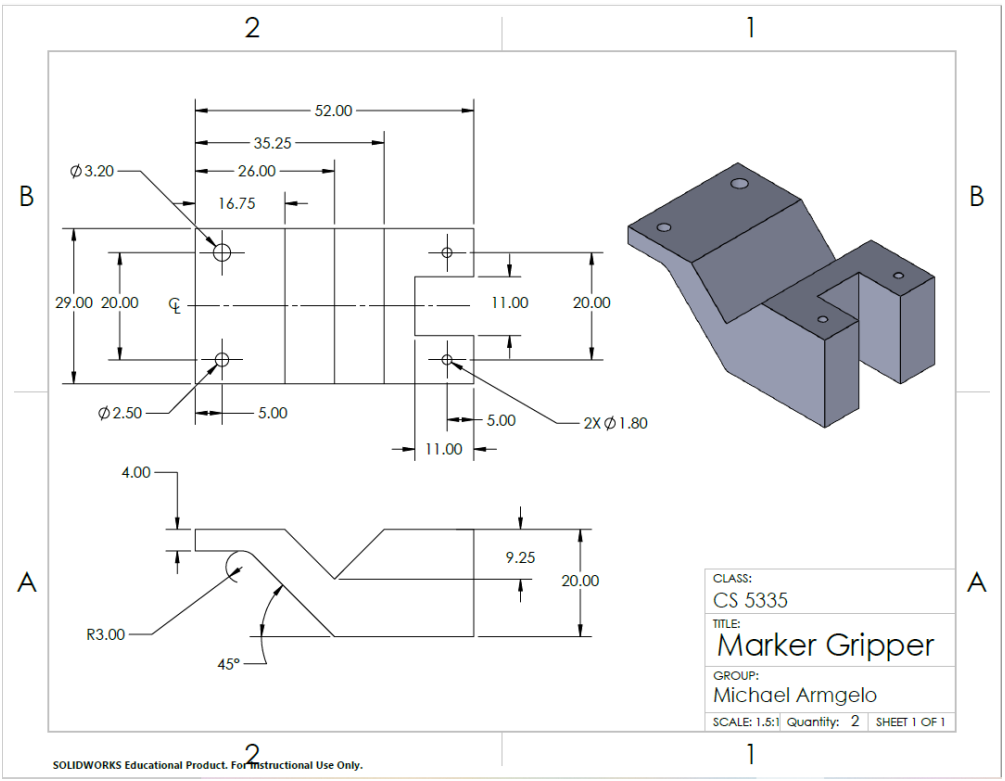


Figure 3: Engineering Drawing of Gripper

2.d Fabrication

The gripper was fabricated using a Creality Ender3 3D printer. The SolidWorks Part File was saved as an STL and printed with standard white PLA plastic using a 0.6mm nozzle diameter, 0.2mm layer height, and 40% infill. Two identical gripper pieces were printed and are shown in Figure 4 below.

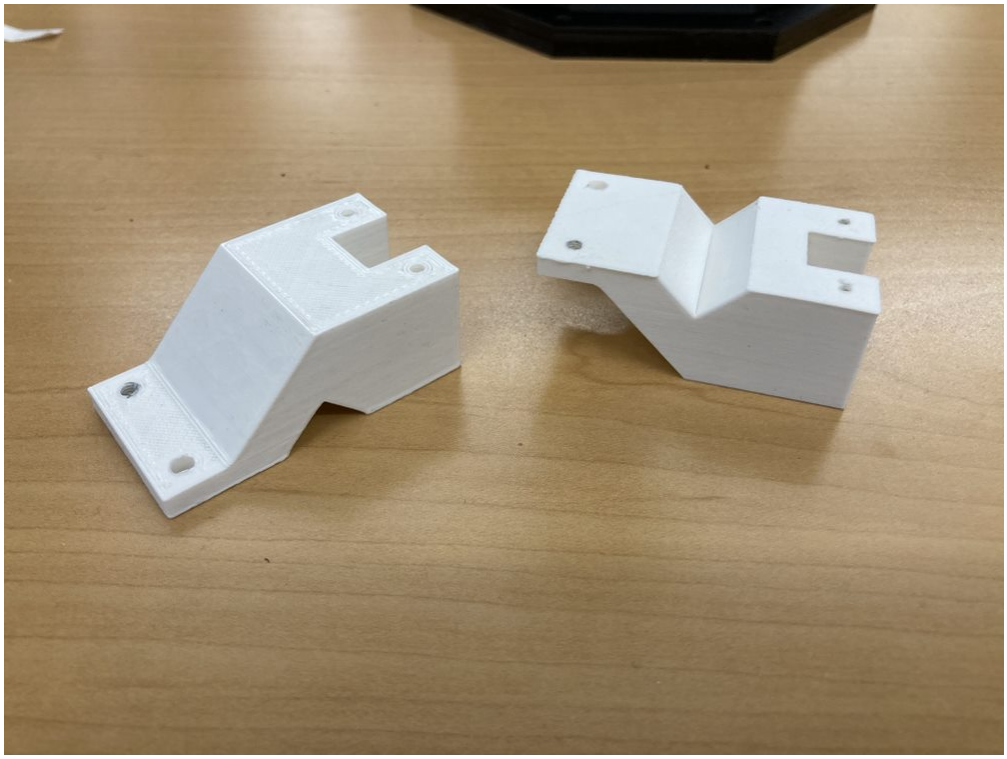


Figure 4: 3D Printed Gripper Parts

Once the pieces were printed, we utilized the foam strip included with the robot. The strip was cut to smaller length pieces so it would fit within the V-slots, and the depth was decreased to 1.5 mm. The back end of the foam slabs is adhesive, which allowed us to attach them directly to the V-slots.

2.e Mounting

We mounted the grippers by first connecting them to the carriages. The carriages were set to their maximum distance apart and both grippers were connected to the sides of the carriages with M2 bolts. The marker was then placed between the two gripper pieces and carriages were closed together until the pen was snug. The pen was gripped at a location where the tip was 55mm below the end-effector position along the Z-axis. Once the pen was gripped, the opposite end of the gripper was clamped to ensure it would stay in place. From this point on, no torque is applied to the gripper servo motor, as the clamp is enough to keep the marker in place.

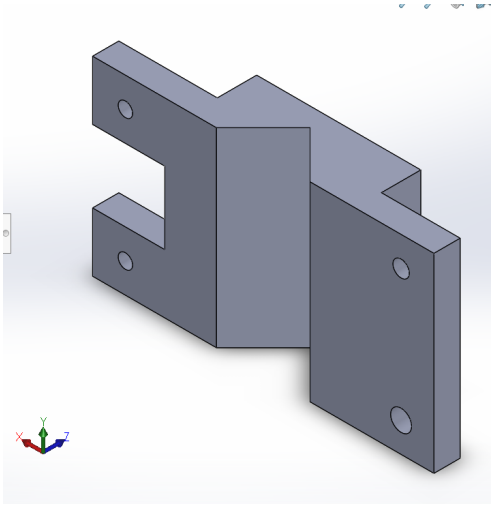
Figure 5: Mounted Gripper with Sharpie

2.f Issues

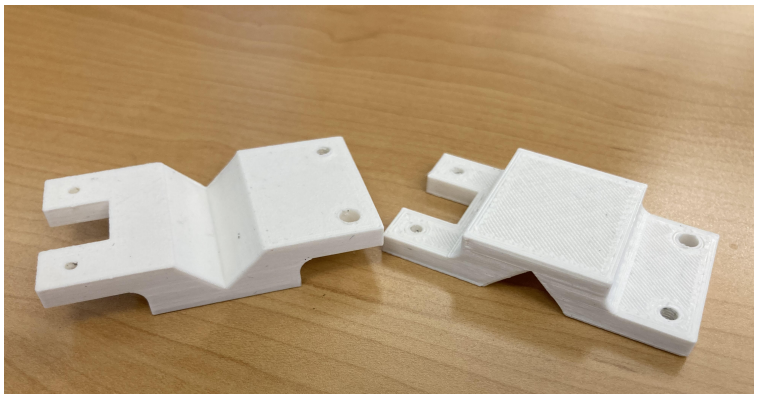
We experienced four major problems with earlier iterations of the gripper design:

1. Fitting the gripper between the carriages.
2. Creating the holes to attach the gripper to the carriages.
3. Holding the marker in place with adequate force.
4. The variable diameter of the markers.

The first part of the gripper design which proved to be challenging was making sure that the marker was held in by the robot, even when subjected to an axial load reaction force by the canvas. With the understanding that the existing flat-faced gripper would not provide the necessary force to hold the pen in place, we designed a two-piece custom gripper with two V-slots to hold the marker. What we did not consider in this early phase, however, was that the minimum distance allowed between the carriage pieces was greater than the greatest cross sectional diameter of the maker. Moreover, the extra 4mm of thickness added to each carriage by the custom gripper design was not enough to compensate for this discrepancy. As a result, the gripper ends could not move close enough together to grip the marker. Figure 6a shows the design for this model in SolidWorks, and Figure 6b shows the print of this piece.



(a) First Gripper Design



(b) Print of First Gripper Design

To address this issue, we increased the thickness of the gripper to 20mm. As we calculated, this successfully closed the gap and allowed the marker to be fully clamped. With this design we made sure 20mm would be less than half of the maximum distance between the carriages and greater than half of the minimum allowable distance between the carriages.

The second roadblock we encountered was mounting the gripper to the carriage pieces. Since the M2 carriage holes were threaded, we planned to bolt the gripper to the carriage with the M2 - 10mm bolt provided. This would entail first inserting the M2 bolt through the gripper hole, then screwing it into the carriage. The holes on the gripper would thus be bigger than 2 mm to more easily fit the bolt. The issue, however, arose due to the increased thickness of the gripper, which we increased to account for the minimum allowable distance between the carriages. At 10 mm, the end of the bolt would not reach the other end of the 20 mm hole of the gripper. Additionally, the increased thickness allowed little room to screw the piece in, once aligned with the carriage. Due to both these issues, we decided to instead bolt the pieces together from the opposite direction, starting from the carriage moving into the gripper. Doing so required a slight modification to the second design. The holes intended to align with the carriages were decreased in size so they could be threaded by the M2 bolt as it was screwed in. Making this change led to the third and final version of the gripper.

With the gripper design finalized and mounted to the robot, we next needed to check if it could successfully hold then pen. Initially, we aimed to evaluate whether holding the marker could be accomplished solely from actuating the carriages. We found however, that although this could hold the pen in place, it faltered if an axial force was applied. We attempted to increase the effort applied by the servo motor which drove the carriages, however even at the full duty cycle, this was unsuccessful. Fortunately, we added clamps to the other end of the gripper as a contingency plan. After the unsuccessful attempts to hold the pen with the carriage forces alone, we deactivated the servo motor and manually moved the carriages as close together as the marker and grippers would allow. Once in tight contact, we used two bolts to clamp the far ends of the gripper pieces together. Following this, the maker remained stable in the grip of the robot, even when we simulated the axial load it could experience when writing on the canvas.

The fourth and final issue we ran into was the variable cross-sectional area of the marker along its major axis. Specifically, the cross-sectional area decreases the further it is from its tip. As discussed in 2.c, this would result in only one plane of contact points between the marker and the V-slot. Consequently, the

marker could be subject to net moments about its non-major axes when subjected to the reaction force of the canvas. Initially, we addressed this issue by screwing the upper clamp tighter than the lower one. Doing so increased the number of contact points between the marker and the V-slot, as the slot between the two gripper pieces would forced bend at the same angle as the pen profile. Although this improved the stability of the marker, it still was not fully secure. This is when we opted to offset the V-slot by 1.5mm and add a foam layer. The foam molded itself to the shape of the pen, drastically increasing the contact area.

3 Physical Environment

3.a Canvas

We had three main types of drawing surfaces ("canvases") in mind when beginning this project. The first is a flat, horizontal area with no occlusions, and we assume there is paper or a whiteboard for us to draw on located on or near the $Z = 0$ plane (in the arm's coordinate system). This was satisfied by simply clamping the arm to the edge of a table, with paper taped onto the table in front of it, as shown in Figure 7.

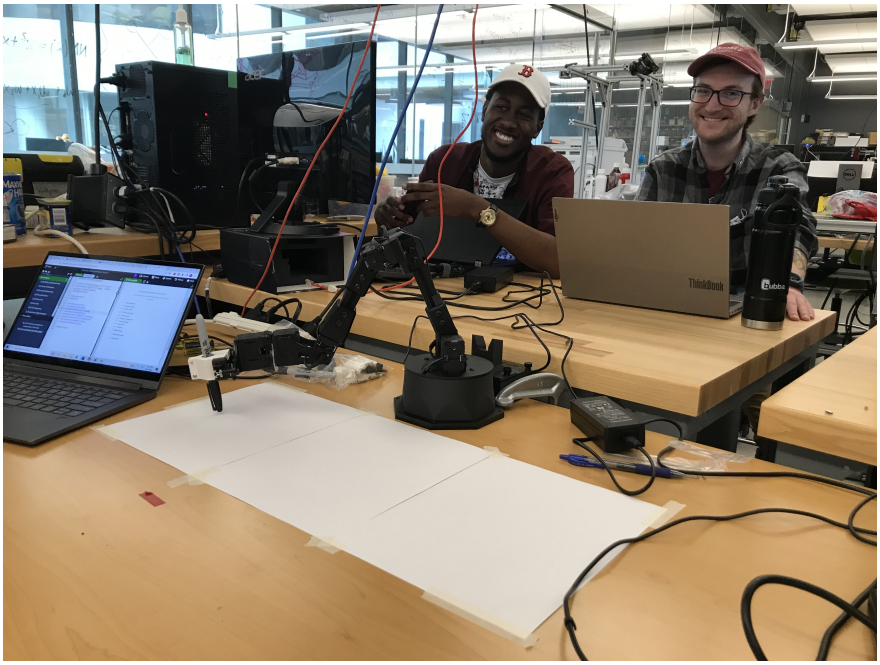


Figure 7: The arm setup to use the flat canvas.

The next canvas is a clear hemisphere which we can use as a domed dry-erase board. We want to be able to draw on both the interior and exterior of the dome, as well as being versatile to different sizes of domes. Our assumption with these canvases is that the center of curvature will be in line with the arm's X-axis; i.e., the center point has some positive X-component, but is at $Y = 0$ and $Z \approx 0$. This is essential, as the method for drawing arcs on curved surfaces cannot move the arm in Y or yaw, which is discussed in further detail in Section 5. The canvas setup to draw on the exterior is shown in Figure 8. Additional canvases on which we have demonstrated good performance are shown in Figure 9.

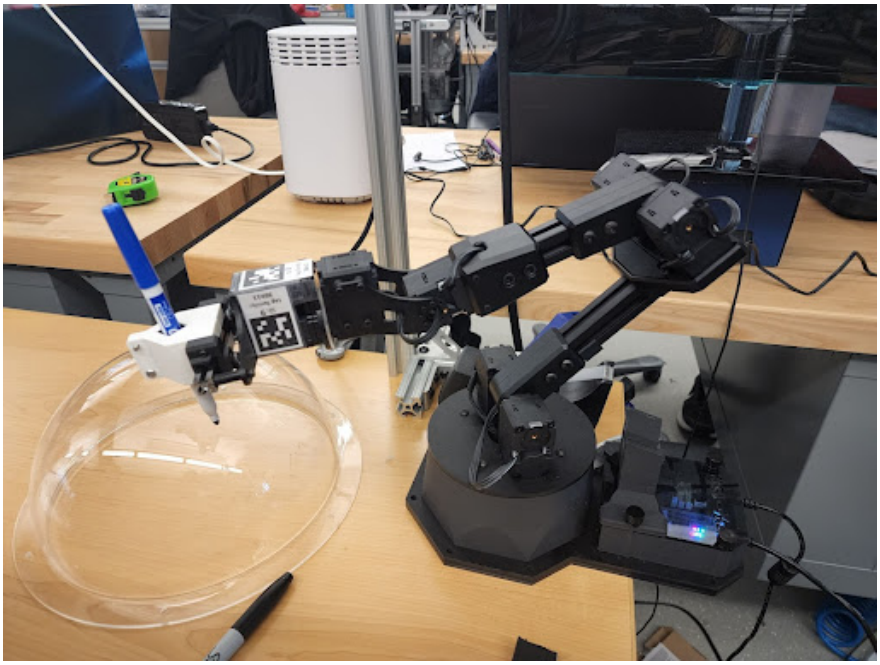
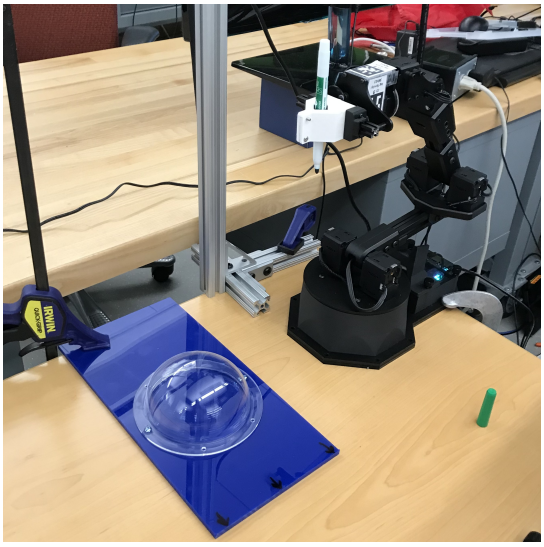
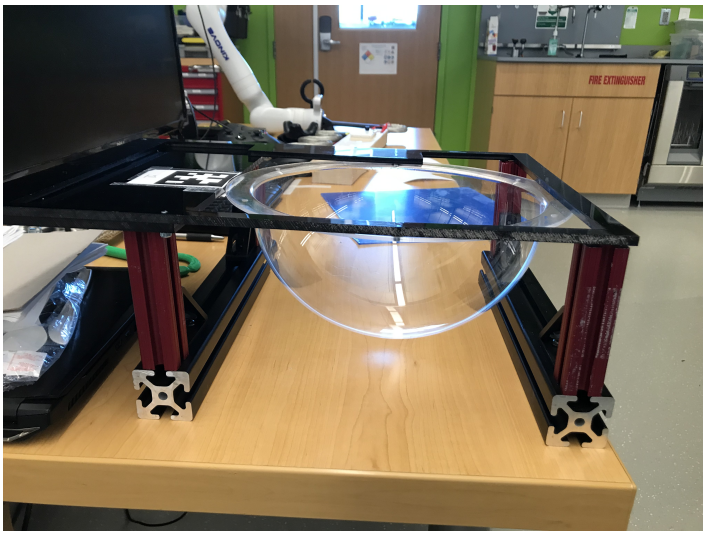


Figure 8: Setup to draw on the curved canvas' convex side.



(a) Convex surface of much smaller hemisphere.



(b) Inner surface of the main hemisphere.

Figure 9: Alternate options for curved canvases.

3.b Camera Mount

To help improve the robustness and reliability of our project, we want to introduce leniency into the positioning of all working parts. To do this, we have an overhead mount to give a birds-eye-view from a RealSense D435 camera. We include AprilTags on the end-effector, at the base of the robot, and on the canvas itself. Thus, we need not measure the distance of the center of the domed canvas from the arm, as we can simply compute the transforms between the arm frame and canvas frames via AprilTag detections.

Detecting the tags on the end effector allows us to get a sense of how accurately the arm is following the commands given to it. We know there is a lot of slop in the arm, especially as the end effector gets lower or far away from the base, so between commanding trajectories, we can detect the true position and command small relative corrections to ensure the overall desired trajectory is followed fairly well.

(a) RealSense D435 camera facing downward, mounted with 80/20 aluminum extrusion.

(b) Camera's view of the environment, with detected AprilTags highlighted.

3.c Issues

A problem with this flat canvas is that the actual reachable area by the arm is quite small. We only want to draw on the paper in front of the robot, which limits the angle to about a 60 degree horizontal field of view (HFOV). Additionally, because of the way the pen is mounted, we cannot draw within a radius of about 15 centimeters from the major (yaw) axis of the arm base, since that would cause part of the arm to intersect with the base. Our effective maximum radius is also less than the technical reachable workspace, since reaching beyond $\sim$ 35 to 40 centimeters causes the shoulder or elbow joint motors to reach maximum torque just from the weight of the arm, and collapse. When this happens, the Dynamixel servos blink red, and the arm requires a power cycle for them to work again. We posit that a better use of the flat canvas could be to have it slanted up (away from the robot) so that when the arm is reaching far, it also has a high-enough Z component that it can support its own weight. The most extreme version of this would be a completely vertical canvas, which would keep the majority of the arm's weight over its own base at all times, and likely improve the effective area of our drawing space.

We have similar problems when drawing on the curved canvases due to the possible pitch range depending heavily on the end-effector's position. As a result, we can generally only draw on the far side of the convex domes and the near side of the concave dome, where the surface normal has no component pointing back towards the arm. On the larger dome, even if the pitch weren't a big bottleneck, we have the same issue as with the flat canvas in that the dome is very large, and must be very close to the arm's base for the far side to be reachable by the pen tip. We would be able to create more interesting drawings if the dome could be mounted overhead of the arm, centered above its base, but that's not very feasible for the scope of this project.

4 Virtual Environment

A full setup guide with all necessary packages and commands is included in the README of our GitHub repository. This can be viewed here, and its components will only be discussed at a high level in this section.

4.a Interbotix Workspace Setup

Interbotix, the creator of the arm we're using, provides a full ROS architecture for getting started working with the arm. This comes in the form of a ROS workspace, `interbotix_ws`, which should be installed to the home directory. This workspace on its own allows for manual control of the joints with sliders in `rviz`, commanding of goal poses, and recording/replay of motion with the arm, to name a few.

After installing the default Interbotix workspace, we clone our MichelARMgelo repository into the workspace's `src` folder; it will be treated as a directory of several ROS packages. Among these packages are the AprilTag library and ROS wrapper, as well as three custom nodes that will be discussed in Section 4.c. Run `catkin_make_isolated` in the workspace, and everything should be built. Connect the arm to power and to the laptop's USB to get started.

4.b Python-ROS Interface to Control the Arm

The X-Series arms by Trossen Robotics supply a Python interface built on top of ROS. This includes several functions that utilize inverse kinematics to move the robot's end effector to a desired position or pose. More specifically, these functions work by commanding the correct joint angles to each motor that will place the end effector in its desired location, given that this location is in the configuration space. The desired joint angles commanded by these functions are set through controlling the effort applied by each servo.

The two functions we used for this project were:

1. `set_ee_pose_components`
2. `set_ee_cartesian_trajectory`

The former, `set_ee_pose_components`, has six inputs for the `x`, `y`, `z`, `roll`, `pitch`, and `yaw` of the end effector. These inputs specify these variables in the global frame of reference. If the final end effector pose is within the configuration space, each joint will move to an orientation that reaches this final position. Using this method, the end effector will follow a valid but unspecified curved path to reach its final state.

The latter, `set_ee_cartesian_trajectory`, allows the user to input the desired relative change in `x`, `y`, `z`, `roll`, `pitch`, and `yaw` from the end effector's current pose. These coordinates are set with respect to the global frame. The function creates a series of evenly spaced waypoints by linearly interpolating between the current pose and the desired pose (designated by the current pose added to the change in each variable that the user inputs). The PincherX 150 is is a 5-DOF arm with only one degree of rotation about the global Z-axis. Given this feature, and the fact that the three subsequent joints rotate about an axis parallel to the first joint's Y-axis, the function cannot accept a change in `y` or `yaw` for the PincherX 150; the small incremental changes between waypoints would be impossible for the arm to achieve, so the function returns an error if there is an attempt to do so. Therefore, in the case of our project, when using this function we can only send four inputs for the change in end effector pose: `x`, `z`, `roll`, and `pitch`.

This `set_ee_cartesian_trajectory` function also has four timing-related inputs:

- `moving_time` determines the duration of the trajectory, meaning the amount of time it takes for the end effector to change its pose by the amounts specified in the four previous inputs.
- `wp_period` designates the amount of time between waypoints. Together, these first two parameters determine how many waypoints will be set between the initial and final pose.
- `wp_moving_time` indicates the time the end effector should take moving between one waypoint and the next.
- `accel_time` indicates the acceleration it should use when moving between waypoints. When these latter two values are low, the end effector will more accurately follow each waypoint, but will be more jerky while doing so. High values will give a smoother overall path that may or may not exactly hit each waypoint on the path.

There are also a couple of functions to open and close the gripper, but we cannot specify a position to go to; the only options are fully open or fully closed. We can specify the effort at which the motors will attempt to pursue their desired position (either fully open or fully closed), but we decided to just disable the gripper motors' torque, and rely on the clamp screws on our custom gripper to hold the pen in place; this proved to be more than sufficient.

4.c ROS Architecture

After demonstrating our ability to control the arm in nontrivial ways, we moved from simple python scripts to a series of ROS nodes that each handle different aspects of our pipeline. The system overview is shown in Figure 11.

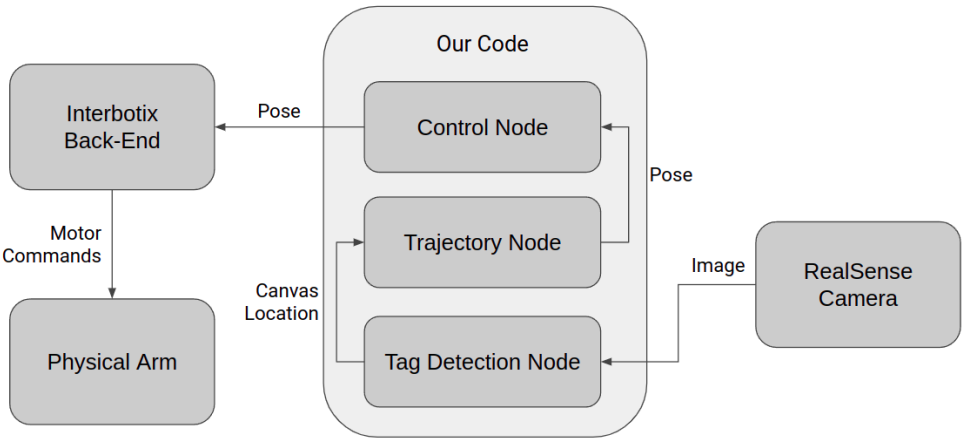


Figure 11: System overview diagram showing the general flow of information.

To simplify the process of running everything for our project, we have made a couple of ROS launch files. The first step is always to launch the Interbotix built-in file to get the arm ready to receive commands, and then we can launch one or both of the following.

1. `control_arm.launch` starts up our two nodes related to actually moving the arm along trajectories. This takes a command line parameter `mode`, which will specify what trajectories will be used.

2. `tag_detection.launch` starts up the RealSense camera driver, the AprilTag ROS wrapper, and our tag detection node. This launch file is also where we specify the static transforms between AprilTags in our environment.

4.c.i Control Node

This node subscribes to poses published by our other nodes, checks them to ensure all points are within the constraints of both the arm's reachable area and our defined workspace, and then commands the robot in a way very similar to how we did in our original scripts, using the two functions described in Section 4.b. All constraints are set via a `.txt` file in the `control_pkg/config` directory of our repository.

In the future, we would like to also create and send trajectories to all joints individually, doing the inverse kinematics on our end instead of letting Interbotix handle it. This will allow much more freedom in the kinds of movements we can do, but will be nontrivial to implement.

4.c.ii Trajectory Processing Node

This node creates or reads a trajectory, and publishes it one point at a time for the control node to receive. The `mode` command line argument is used here to determine the source of trajectories. Some of the `mode` options are:

- Read a trajectory from a file.
- Prompt the user for one pose at a time to freely control the arm.
- Move the arm to random poses within the workspace every few seconds.
- Draw a line on the flat canvas.
- Draw a random line on the flat canvas every few seconds.
- Draw a square on the flat canvas.
- Draw a circle on the flat canvas.
- Draw a circle with a random size/location every few seconds.
- Draw an arc on the convex canvas.
- Draw a triangle on the convex canvas.
- Draw a triangle on the smaller convex canvas.
- Draw an arc on the concave canvas.

4.c.iii Tag Detection Node

Images from the camera are used by the AprilTag ROS package to publish all tags detected in the scene to the `/tag_detections` topic. Our node then subscribes to these messages, computes transforms we'd like to know, and issues resulting information for the other nodes to use. This process is described in detail in Section 6

5 Trajectory Planning

5.a Initial Python Scripts

We began by creating simple python scripts to interact with the interbotix_ws by initializing the bot object and sending it desired poses. We're letting Interbotix handle the inverse kinematics for at this point in time by using the previously discussed `set_ee_pose_components` function. This allows full control of `x`, `y`, `z`, `roll`, and `pitch`. We created a way for the user to input either one pose at a time, or a series of poses that the arm will then execute. To simplify extensive testing, we also made a mode that will read a trajectory from a file, to save us typing in all the points each time.

After demonstrating functionality from a simple python script, we went on to implement and use the ROS nodes described in Section 4.c.

5.b Drawing on a Flat Canvas

Something we immediately noted is that by using the `set_ee_pose_components` function, we hand over not only the responsibility of computing the inverse kinematics to obtain all joint trajectories, but also the ability to control things like the path taken between points and the delays in the process. For example, to draw a straight line on the plane, we cannot simply give two points, as the shortest path in configuration space will result in a curved line being drawn on the paper. In this specific case of drawing a single straight line, we can instead use the `set_ee_cartesian_trajectory` function, but we have a problem again when we want to draw any other shape.

With the goal of drawing a fairly believable circle on the paper, we figured we could parameterize the circumference of the circle into as many points as desired to achieve some granularity, and send each point one at a time to the arm. This does work okay, but has two problems: firstly, the arm pauses at every single point, meaning it moves painfully slower the more points we include in the circle; secondly, this intermittent pausing and slow movement means the arm is constantly struggling to overcome the friction of the writing surface; rather than moving in one smooth motion, the pen tip remains stationary until the arm provides a sufficient horizontal force component to drag the pen tip to the next point. This effect was somewhat reduced by raising the drawing height to lessen the magnitude of the writing surface's reactive normal force, and by switching from drawing with a Sharpie marker on paper to drawing with a dry-erase marker on a smooth dry-erase board, but we are ultimately limited on what can be achieved.

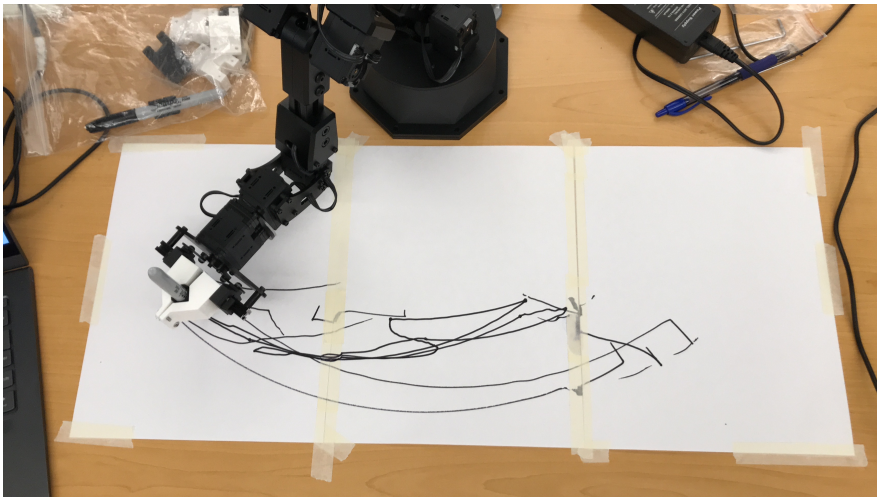


Figure 12: The arm connecting random points on the flat canvas.

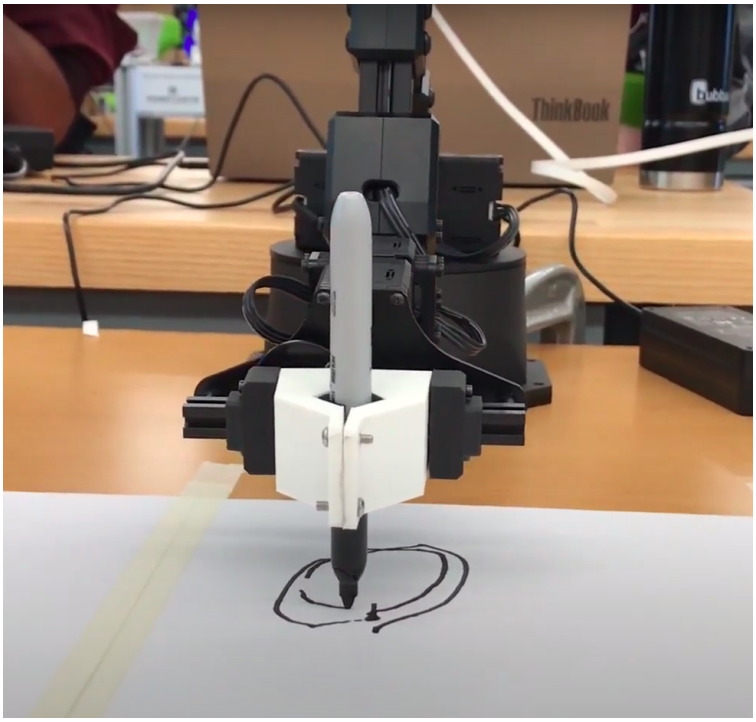


Figure 13: The arm drawing a circle parametrized by points every $\frac{\pi}{6}$ radians, inside a larger circle with points only every $\frac{\pi}{3}$ radians.

5.c Drawing on a Convex Hemisphere

The somewhat-manageable annoyances of drawing on flat surfaces become an unavoidable problem when drawing on a curved surface. We want to draw arcs connecting two points on the outer surface of our dome (which we shall also refer to as a "sphere" for geometric convenience). If we continue in our current behavior, the arm cannot achieve this, as our options at the moment are (a) allow Interbotix to decide the shortest path in configuration space, or (b) draw a straight line in cartesian space. Neither of these will work, since

the shortest straight line path will intersect with the sphere, and the path Interbotix chooses may very well intersect with the sphere or lift the pen off the surface. We decided the best way to handle this would be to create a copy of the `set_ee_cartesian_trajectory` function that we'll call `set_ee_arc_trajectory`, and modify it to take the geometry of the sphere into account when drawing paths.

The original function works by taking the desired displacements $\mathrm{d}x$, $\mathrm{d}z$, $\mathrm{d}r$, and $\mathrm{d}p$, and dividing them by the desired number of points, N. To interpolate a path with N points, then, they iterate i from 1 to N, and add a new point with components:

$$x = x_0 + \frac{i}{N}\,\mathrm{d}\,x$$
$$z = z_0 + \frac{i}{N}\,\mathrm{d}\,z$$
$$roll = r_0 + \frac{i}{N}\,\mathrm{d}\,r$$
$$pitch = p_0 + \frac{i}{N}\,\mathrm{d}\,p$$

This is packaged back together into an SE(3) transformation matrix, which is then sent directly to command the arm after the full trajectory is determined. The advantage of this method is that the arm will not pause at each point, and if we simply change these update equations, we can create a way to draw smooth arcs on the dome.

Since we cannot control `y` or `yaw`, as previously discussed, it seems we're limited to the first joint's current XZ-plane at the time the function is called. Working within these constraints then, we can see that our arc is parametrized by some change in `x`, `z`, and `pitch`. With this in mind, we can derive a simple equation to follow an arc along the surface of the sphere.

$$x = x_0 + \mathrm{d}\,x\left(1 - \cos\left(\frac{i\pi}{2N}\right)\right)$$
$$z = z_0 + \mathrm{d}\,z \cdot \sin\left(\frac{i\pi}{2N}\right)$$
$$roll = r_0$$
$$pitch = p_0 + \frac{i}{N}\,\mathrm{d}\,p$$

This is functional, and allows us to draw an arc along the great circle coplanar with the arm's XZ-plane.

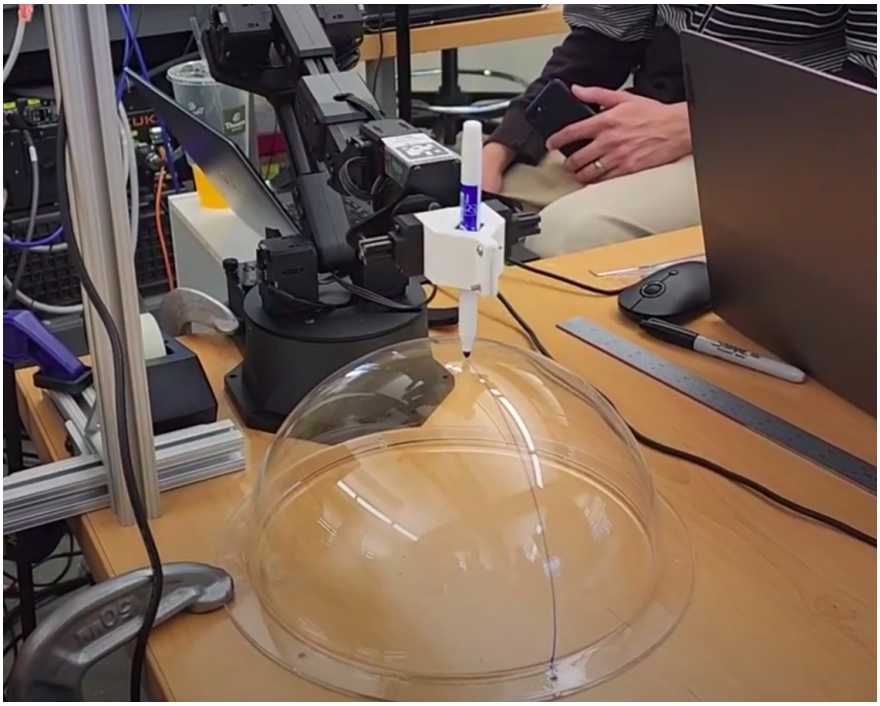


Figure 14: The arm drawing an arc that remains within its plane.

In order to draw outside this plane, our only option is to utilize the `roll` component. If we change the `roll`, however, the pen tip will be lifted off the drawing surface; to compensate for this, the `z` component has a `roll`-dependent term appended to it.

$$x = x_0 + \mathrm{d}\,x \left(1 - \cos\left(\frac{i\pi}{2N}\right)\right)$$
$$z = z_0 + \mathrm{d}\,z \cdot \sin\left(\frac{i\pi}{2N}\right) - \mathrm{PEN\_OFFSET} \cdot \left(1 - \cos\left(\frac{i}{N}\,\mathrm{d}\,r\right)\right)$$
$$roll = r_0 + \frac{i}{N}\,\mathrm{d}\,r$$
$$pitch = p_0 + \frac{i}{N}\,\mathrm{d}\,p$$

where PEN_OFFSET $= 0.055$ is the distance of the pen tip from the presumed end-effector position in the arm's URDF, along the pen's major axis. This function now allows us to draw arcs off-center on the dome.

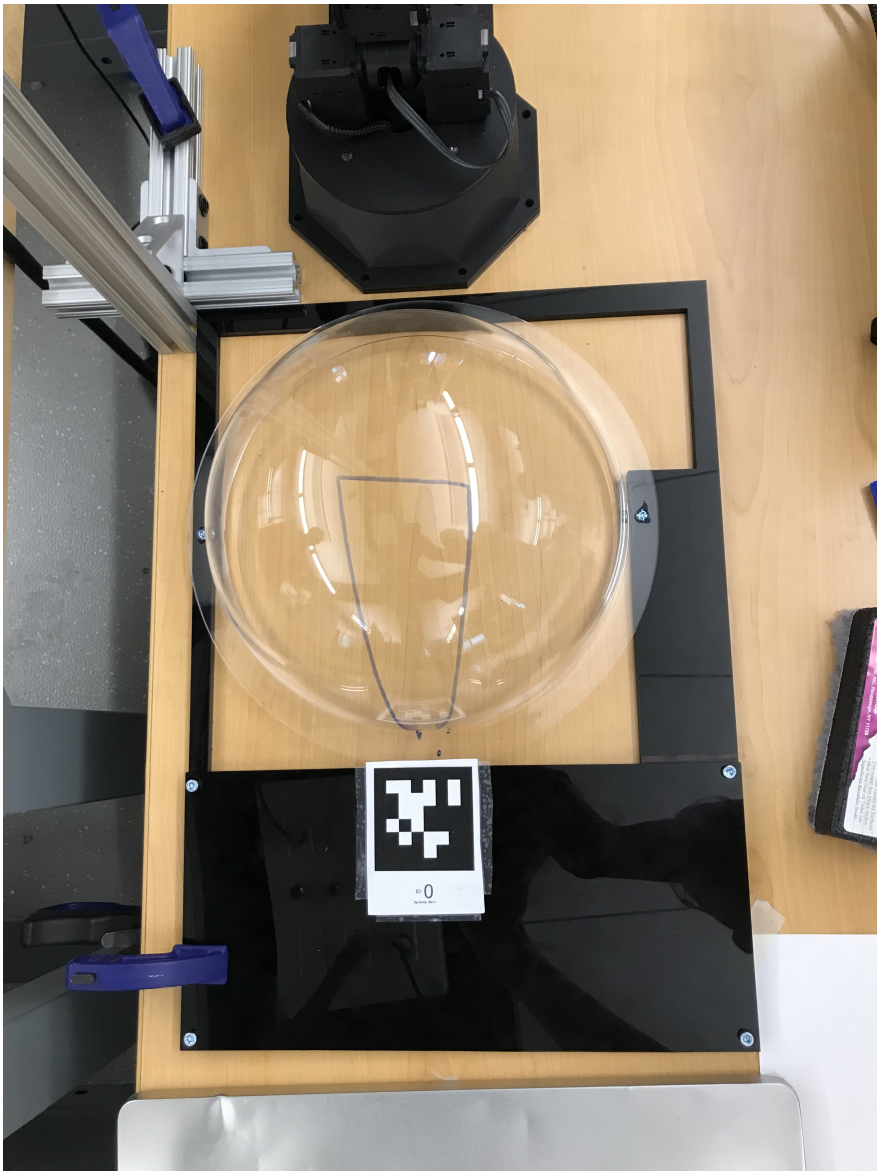


Figure 15: The arm drawing three arcs that close in a triangle on the surface of the sphere.

This formulation is also versatile to the size of the dome itself, and by simply changing the desired dx and dz components, we can draw on our other, much smaller dome canvas.

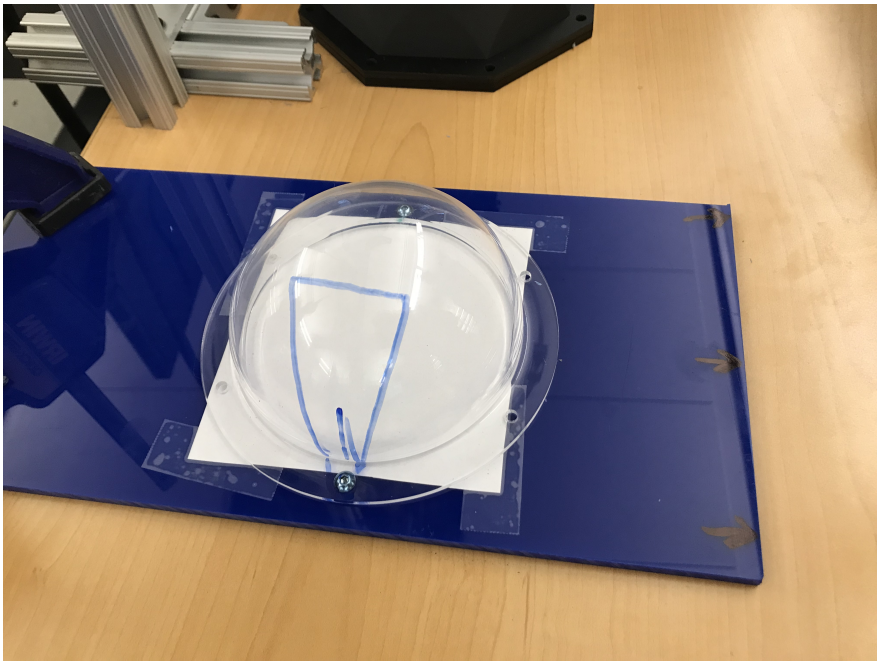


Figure 16: The arm drawing a triangle on the smaller dome.

We attempted to improve this function further by utilizing some properties of geodesics, the shortest path between points on a spherical topology. This is fairly straightforward with the following process:

1. Interpolate the straight-line cartesian path between the start and end points.
2. For each point on this interpolation, compute the vector from the center of the sphere to that point.
3. Divide by the magnitude of this vector to obtain the unit vector storing the direction of the point from the center.
4. Project the unit vector to the surface of the sphere by multiplying it by the sphere's radius.
5. Add this extended vector to the center point to obtain its cartesian coordinates.

This will, geometrically, give us a perfect path along the surface of the sphere that takes the shortest route within the plane defined by the center and the start and end points. However, it does not work in our application. The reason for this is that the arm's coordinate system is deeply flawed, due to slop in the arm, failure to correct for the increasing gravitational torque as it reaches further, and/or a poorly tuned PID controller for the joint motors. As an example of this, for drawing the main arc on the large dome, we commanded a starting position of $(0.4, 0.0, 0.1)$. This already shows error, since a `z` of 0.1 should be 10 centimeters off the table, not touching it. Even worse is that our commands to draw this arc, which we'd expect would be $|\,\mathrm{d}\,x| = |\,\mathrm{d}\,z| = R$ (where R is the radius of the sphere) are actually $\mathrm{d}\,x = -0.19$ and $\mathrm{d}\,z = 0.075$, while the true radius is $R \approx 0.115$ m. Because of this gross failure to adhere to a sensible coordinate system, we cannot generalize and rely on the geometry of our environment, since the true position of the arm and the path it takes are barely recognizable compared to our expectations.

5.d Returning to the Flat Canvas

After achieving the task of drawing on the convex canvas, we have a much better understanding of how to manipulate the end effector to our will. Now that we've written a function to draw arcs, we can use this on the flat surface, even to draw straight lines. (An arc need not change in three or even two components for the equations to be valid.)

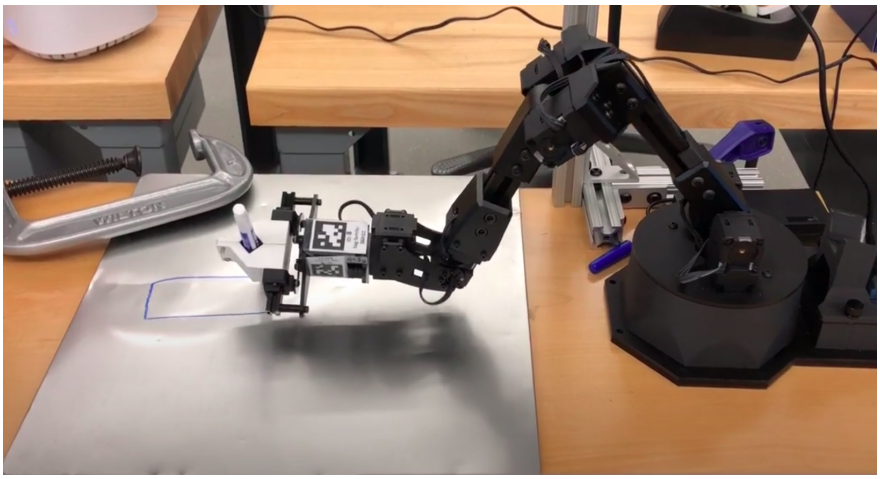


Figure 17: Rectangle drawn on a flat whiteboard.

5.e Drawing on a Concave Hemisphere

We can now move on to adapt our arc trajectory function to draw concave rather than convex arcs. This turned out to be possible with a change only to our interpolation function in `set_ee_arc_trajectory`. We copied this function to create `set_ee_inner_arc_trajectory`, and changed our equation formulation to the following:

$$x = x_0 + \mathrm{d}\,x \cdot \sin\left(\frac{i\pi}{2N}\right)$$

$$z = z_0 + \mathrm{d}\,z\left(1 - \cos\left(\frac{i\pi}{2N}\right)\right)$$

$$roll = r_0$$

$$pitch = p_0 + \frac{i}{N}\,\mathrm{d}\,p$$

We also need to change our range for `pitch`, since at the flat part of the arc, the end effector cannot be normal to the surface, as that would cause part of the arm to intersect with a higher section of the dome. With these changes, we're able to draw on the inner surface.

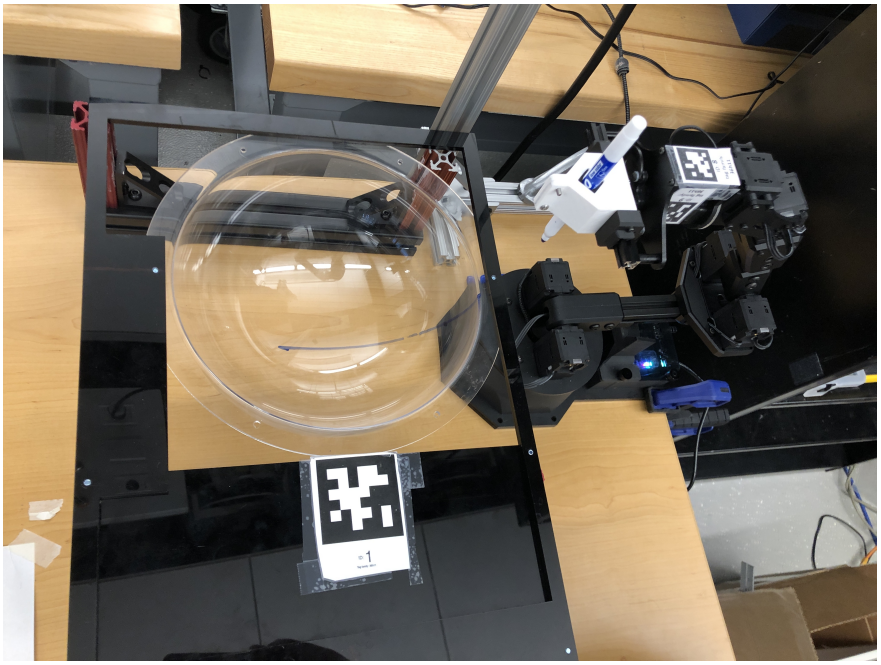


Figure 18: The arm drawing an arc on the inner surface of the dome.

We do have a more limited range of motion inside than outside the dome, as the horizontal aluminum channel in the gripper assembly is fairly wide, preventing how far the pen can roll to one side before colliding with the surface.

6 AprilTag Detection

Our tag detection node subscribes to the `/tag_detections` topic that AprilTag ROS uses to report the poses of all detected tags in the camera's frame. We know to expect the following tags in the environment:

- Tag 0: on the convex canvas, with known transform T_{0C} to the center of curvature.
- Tag 1: on the concave canvas, with known transform T_{1C} to the center of curvature.
- Tag 2: at the base, with known transform T_{2A} to arm's coordinate frame.
- Tag 8 and Tag 9: on the end effector of the arm, with known transform relative to each other T_{89} and to the pen tip T_{8P}.

When we detect Tag 2 and either Tag 0 or Tag 1, we can set the position of the center of the dome in the arm's coordinate frame. Let the RealSense camera's frame be represented by R.

$$T_{CA} = T_{C0} \cdot T_{0R} \cdot T_{R2} \cdot T_{2A}$$

This allows us to calibrate our parameters for drawing arcs on the dome and allow leniency in the environment setup. Since the camera's frame is used only as an intermediary, its pose is unimportant as long as all tags are visible. This makes setup very simple.

The tags can also be used to localize the end effector. When we detect Tag 2 and either Tag 8 or Tag 9, we can find the pose of the pen tip in the arm's coordinate frame. Note that the transform T_{8P} depends on the current `roll` parameter of the end effector, and thus must be recalculated using the current EE pose.

$$T_{PA} = T_{P8} \cdot T_{8R} \cdot T_{R2} \cdot T_{2A}$$

After determining the true position of the end effector, we can use its difference from the commanded pose to issue a relative correction command, and thus keep the arm from drifting further from desired with consecutive trajectories.

6.a Issues

Due to our realization of the arm's poor adherence to a strict coordinate system, as we discussed at the end of Section 5.c, the usefulness of the vision component of our project has been very limited. Ideally, we'd like to detect the canvas via either an AprilTag or by performing RANSAC on the pointcloud from the depth camera, and then use this knowledge of the canvas to inform where we draw; i.e., we could find the center of the hemisphere and draw on the surface. However, since the arm controls are in actuality quite divergent from the virtual setup, differing in each axis, and with nonlinear error, we cannot rely on the geometry of our setup at all. Knowing the position of the center of the sphere and its radius does not help us to draw on its surface, whether we detect these parameters from the camera or measure them directly by hand. Due to this fact, the generality of our project is also limited, as all trajectories must be manually tuned for an exact setup of the canvas and arm.

7 Discussion

7.a Evaluation of Performance

Our group successfully completed our primary objective: programming the PincherX 150 arm to use a marker to draw on a canvas. First, we designed a custom end effector gripper design to mount to the arm and hold the marker (Section 2). This was necessary to ensure the marker was secure in the robot's grip as it drew. Then we made several different canvases for the arm to draw on: a flat whiteboard, large convex hemisphere, large concave hemisphere, and a small convex hemisphere (Section 3.a). We next designed several trajectories for the pen end effector to follow in order to draw our desired shapes on the canvases. This entailed setting up the Interbotix Workspace and writing the `traj_processing_node` and `control_node` to form and command these trajectories to the end effector's `x`, `y`, `z`, `roll`, and `pitch`. (Section 4). We did so here by calling the functions `set_ee_pose_components` and `set_ee_cartesian_trajectory`, in addition to altering the latter to write our own function `set_ee_arc_trajectory`, which interpolates the waypoints over an arc as opposed to doing so linearly (Section 5). Finally, we set up the RealSense D435 Camera and AprilTags in the scene, and created ROS node `tag_detections_pkg` to subscribe to `tag_defections` and compute transforms (Section 6).

With the accomplishments above, we successfully met four of our six sub-goals:

1. We designed a custom gripper which successfully held the pen in place, even when subjected to high axial loads.

2. With the function `set_ee_pose_components` and `set_ee_cartesian_trajectory` we drew pre-planned shapes such as an arc, straight line, circle, and square on a flat surface.

3. By writing the `set_ee_arc_trajectory` function, with the mathematical derivations required to parameterize the arcs, we were able to complete our third goal of drawing shapes on the exterior of a domed surface. Here the arm successfully drew both smooth curves and a triangle on the domes.

4. Modifying the prior function to create `set_ee_inner_arc_trajectory` allowed us to draw on the inner surface of the dome as well, completing our fourth goal.

Though we did not complete the sixth sub-goal of utilizing the camera to localize the end effector pose (see section 6.a, we set up the ROS architecture necessary to complete this in the future, and it will rely on improvements to the realistic controllability of the arm. Overall, we are satisfied with the number of milestones we reached and believe the strategies utilized to complete these would set the groundwork for the two we did not meet, in addition to any other goals we may want to pursue in the future. These are described in detail in Section 7.c. A video montage that includes several demonstrations of our progress and final product is posted to YouTube here.

7.b Summary of Issues and What We Learned

7.b.i Gripper

We encountered our first set of issues when designing the gripper. These roadblocks are described extensively in Section 2.f. In short, the gripper achieved its two primary functions, mounting to the arm and holding the pen even under axial loads, after our third design iteration. The crux of our issues stemmed from unforeseen design requirements that we did not plan ahead for, such as the minimum allowable distance between the carriages on the end effector. Fortunately, design iterating and prototyping were not foreign concepts to our group. We all were experienced with SolidWorks and additive manufacturing, the two tools required to create the custom gripper. Consequently, there was not a substantial amount of time spent learning new software here. We did however, learn more about how to make an effective clamp. V-Slots are often the preferred method over a half circle. The former guarantees at least two points of contact between the item and the clamp. While the latter would have the same profile as the marker, it would be difficult to get match its diameter exactly to the clamped item, leading to a less effective singular point of contact. To future CS 5335 students who want to create a custom gripper, we would recommend reading Interbotix's documentation on custom grippers. This includes an engineering drawing of the carriages so the designer can ensure their model will fit to the arm. The assembly model of the arm is also provided, and can be a great tool to visualize how one's gripper model appears and functions when mated to the rest of the robot.

7.b.ii Canvas

In the early stages of drawing with the arm, we found several issues with the canvas setup. As described in Section 3.c, our choice for the canvas orientations potentially limited the art the arm could achieve. In more detail, the flat canvas, which was used for many of the early trials, was at a lower point than the first joint of the arm. Due to this location, the arm only had a small set of configurations where the marker's tip was coincident to the plane of the canvas, limiting what we could draw. This was compounded by our discovery that the arm struggled reach far distances from its base, as the moment created from its weight would exceed the maximum achievable torque of the motors.

From this challenge we learned the importance of evaluating the workspace of the arm before ideating goals for it to achieve. From doing so, we may have determined early on that a vertical or diagonal canvas could have increased the number of viable locations for the marker tip to reach. Additionally, we would have determined that an extra degree of freedom would have been more appropriate given the shapes we wanted to illustrate. This would have been particularly useful for the sphere canvas. Although killing each motor and manually moving the robot to different configurations is illuminating, it does not paint a complete picture, however. Attempting to draw the trajectories also demonstrated to the team that just because a robot may be able to reach a certain configuration does not mean the joint torques or the path planner support it. Consequently, there were points on the canvas we thought were feasible, but in actuality were unachievable. To future teams, we recommend evaluating the robot's complete workspace, both through viable configurations and what is achievable through the motor torques in order to compose a realistic plan for how the arm may interact with its environment.

7.b.iii Virtual Environment

It took some time for us to setup and understand the extent to which we have control over the arm; after reaching a certain point, however, we did not have any significant issue sending commands to the arm and understanding its response, whether it be an error or an unintended motion. This was likely due to the fact that every member of our team had previous experience using ROS. Most real issues with this section relate more to specific trajectories, and will be discussed next. We learned a lot about the different ways to control the arm, and what its limitations are. To a future group using a similar robot arm from Interbotix, we would highly advise that they consider the possible range of motion of the arm as it compares to their task. For us, in hindsight a 6-DOF arm that has local `yaw` control would have been very nice, but we did not consider that when requesting our arm model. We would also recommend having a group discussion about the advantages of using ROS nodes, and whether they are necessary to achieve the team's desired outcomes. Our group implemented ROS nodes to command the trajectories, but with hindsight a script could have served as an easier alternative.

7.b.iv Trajectory Planning

In the `set_ee_cartesian_trajectory` function, Interbotix imposes a restriction on the allowable commands, such that any change in `y` or `yaw` will throw an error if the arm model being used has fewer than 6 degrees of freedom. Even bypassing this restriction by editing this code does not permit successful trajectory planning. For our PincherX 150 arm with 5-DOF, then, we were not able to utilize the Y-axis other than when commanding a single pose for the arm to go to; recall that this isn't desired as it will stop at each point and thus smooth trajectories are impossible. We know, however, that many of the sideways motions we wanted to achieve are absolutely possible with this arm, but we simply have no way to command such trajectories smoothly.

We've also discussed in this report how the arm does not accurately adhere to a coordinate system. Its inaccuracy increases the farther the end effector is from the base, as this increases the moment that each motor torque must counteract. This may be due to poor tuning of the motor PIDs, or perhaps simply an unfortunate factor of this particular arm model. In any case, this meant we could manually tweak values such as $\mathrm{d}x$ and $\mathrm{d}z$ to achieve the desired motion, but it required many steps of trying and tweaking values to get the motion we wanted; the final values for many of our paths were very different from reality, such as when drawing an arc on the hemisphere's outer surface, as discussed in Section 5.c.

We would give the advice to a future group to look into these issues at the very start, and spend some time developing a custom controller and working on tuning. It was difficult to have already done work that relied on a reasonably accurate range of motion only to realize that this was not always the case, inciting a need to rework or leave unused some of our code.

7.b.v AprilTag Detection

AprilTag detection on its own had no real issues. We were easily able to observe the entire workspace, and all tags of interest, as well as to compute desired transformations. However, the issues with the trajectory control prevented us from being able to truly utilize the results of this node, as discussed in Section 6.a. For a future group, we would suggest they consider whether vision will really benefit the project, and prioritize things like accurate control that are a prerequisite to the benefits of vision.

7.c Future Steps

7.c.i Trajectory Control

The main takeaway from these issues discussed above is that in the future, if we or another group attempts to improve on this project, it will be necessary to write a custom controller that does the inverse kinematics and commands all joints of the arm directly; this would allow a far greater degree of customization in addition to freedom over how the arm can move. A future project should also attempt to remedy the concerns relating to the commanded pose vs actual pose before continuing with their chosen model of arm, as it would open the gates to better control, more accurate paths, and more compatibility with the camera localization.

7.c.ii More Complex Trajectories

We were able to demonstrate some manually chosen trajectories with success, but with the improved controller just discussed, more complex paths will be able to be executed smoothly. It will then be possible to tackle some of our more lofty initial goals, such as writing alphanumeric characters and replicating seen paths. Initially these could be hard-coded, but the ultimate goal would be to generate the trajectory from some predefined geometries, font files, or images.

An additional interesting goal would be to implement a computer vision module that can take any image, extract from it the main lines, and convert these to trajectories for the arm to follow, allowing any image to be roughly reproduced by the arm.

7.c.iii Vision Implementation

The camera could have been a great boon for the accuracy of our drawings, but we simply could not use it due to the issues with the arm discussed at the end of Section 5.c. With the changes discussed in Section 7.c.i, this will be much more feasible. We'd like to see the AprilTags and/or depth information from the camera used to inform the location of features in the environment. This includes the center of curvature and radius of the hemisphere canvas, as we've discussed, but could also allow us to detect and avoid any obstacles in the workspace, and even to ensure we don't draw off the paper. Vision can be a powerful tool that unlocks many opportunities, and we'd like to see it utilized better in the future.

A Contributors

While everyone on our team contributed to the overall scope and planning of this project, we would like to call out certain aspects of the project that benefited from the particular strengths of each team member:

- **James Tukpah** - Initiated and co-developed the software architecture to plan and execute trajectories; constructed the camera mount.
- **Kevin Robb** - Co-developed the software architecture to plan and execute trajectories; worked on geometric trajectory derivations; set up the camera to perform AprilTag detection.
- **Richard Kaufman** - Fabricated, assembled, and co-designed the gripper; worked on geometric trajectory derivations.
- **Michael Carvajal** - Co-designed the gripper; proposed, procured, and constructed various canvas components; worked on geometric trajectory derivations.

Last Thoughts

Overall, this was a fairly successful project in which we learned a lot, and had fun. Despite some frustrations and setbacks, we're proud of our end product, and were happy to have had the opportunity to choose such a project and borrow a robot arm to make it happen. Thanks to Dr. Lawson L.S. Wong for teaching the CS-5335 course and to Northeastern University for supporting the project materials.