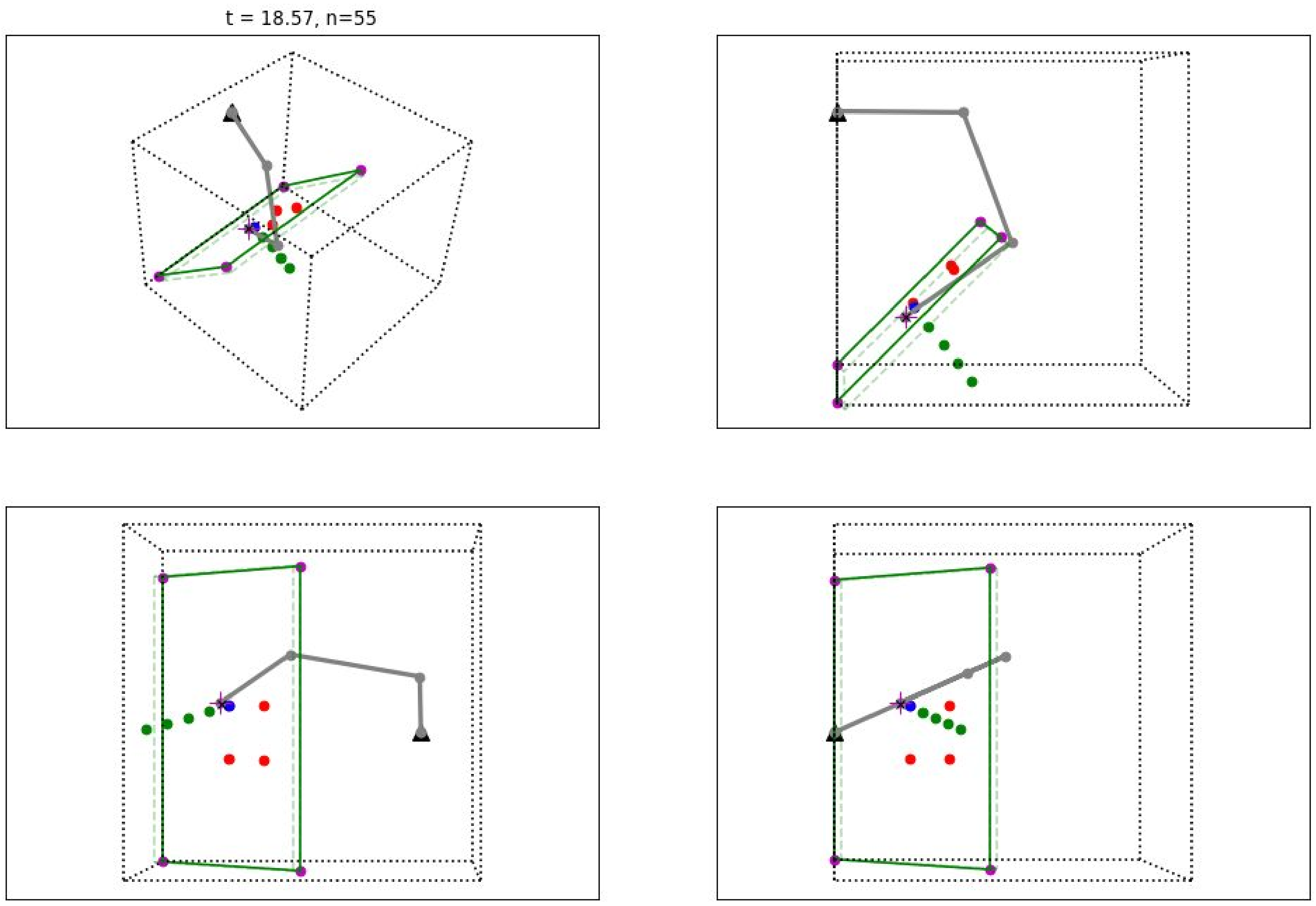
EECS 464: Hands-on Robotics - Project 2 Report

Cyan Team

Vaibhav Bafna, David Chang, Eric Wiener

Winter 2020



Contents

1. Background

1.1 Project Task Specifications

The objective of project 2 was to create a robot arm that could draw a square on a piece of paper as described in the 2020 Project 2 Task Specification. The project was adjusted this semester to be done in a simulation, and the details about the simulation and how to modify the code can be found here.

1.1.1 Simulation

The simulation was designed to model the real properties of a serial manipulator. For instance, motors had current and thermal limits. If a motor exceeded its thermal capacity, then it would shut down. The force of gravity could cause the robot's arm segments to sag, and gearboxes had backlash. Furthermore, because pens drag on paper, the pen stroke on the paper trailed the pen's actual position. Our robot arm was a serial manipulator with only rotational motors/joints (used interchangeably) and had a pen attached at the end of the arm as the end effector. The "results" of each simulation run were a three-dimensional trajectory of the pen and a 2D image showing the drawing results on the paper.

1.1.2 Simulation Parameters

There were three main areas of development - the arm design, the workspace, and events. The "armSpec" was a specification of the arm and described axes orientations', segment lengths, and initial angles. Within this specification, we could add as many segments as we liked and configure them to our desired specifications. We could control the transformation of the workspace to world coordinates by modifying the "Tws2w" transformation matrix. ("armSpec" and "Tws2w" can be found in ./src/myarmsim.py). Finally, we could add our own "Plan" classes and leverage the "arm" motor simulator objects to control the motors. However, we were only allowed to use "getter" and "setter" methods on these objects, which were located in the "MotorModel" class.

1.1.2 P-Day

On P-day, teams ran their simulated robot on a series of ten different paper and square orientations within the workspace, which we refer to as the different arenas. The professor provided the X,Y center coordinates of the square and the scale of the square in paper coordinates. We interpreted the scale as representing the full side length of the square, but the professor later clarified that his intention was for the scale to represent half the side length. He accepted either interpretation of the scale. Three example arenas are shown below in Figure 1. A list of all the paper and square specifications for each arena provided can be found here.

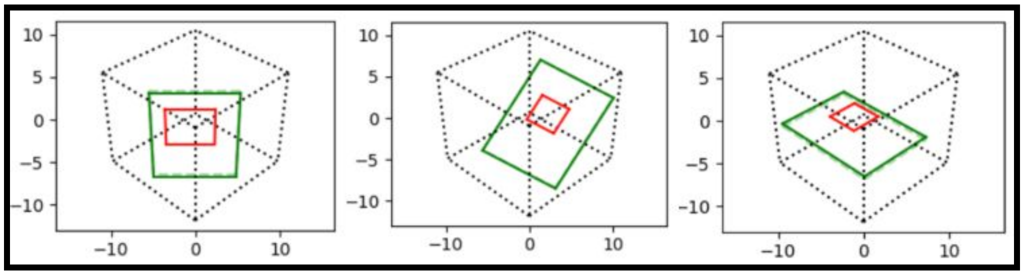


Figure 1. Arenas 0, 1, and 2 (from left to right).

1.1.3 Pass/Fail Requirement

The initial pass/fail requirement was to draw a closed shape, with four identifiable corners, on the paper, on at least two orientations. To draw a shape, the pen had to be pushed into the paper 0 to 10 mm from the front of the paper. Before P-day, the pass/fail requirement was modified so each side must be at least a length of 2 units instead of L*0.2 and you only had to draw a shape meeting the requirements on one orientation of the paper.

Pushing into the page beyond 10 mm from the front did not disqualify you, but only what was drawn in black in the 2D results image counted towards the drawing. We were also allowed to disable gravity for 70% of pass/fail credit.

1.2 Simulated Robot Design

Before beginning the coding of our simulated robot, we proposed a potential design that we presented in our brainstorming presentation and gathered resources to help us understand forward and inverse kinematics, which were important topics for this project.

1.2.1 Arm Design and Workspace Transformation

Our robot arm was composed of three segments each of length 5 units. The first joint, located at the base of the robot arm, rotated about the z axis, while the following two joints each rotated about the y axis. The first and third arm segments had initial angles of 0 degrees, while the second arm segment had an initial angle of 90 degrees. The base of the arm was always at the world origin. We placed the workspace origin, which was the bottom back left corner of the workspace, at the position (0,-5,-10) in world coordinates, as seen in Figure 2.

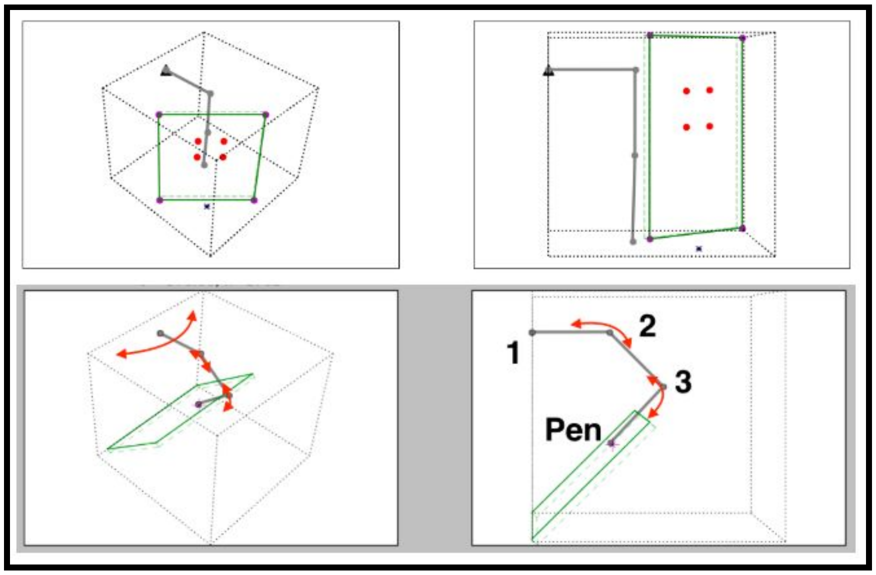


Figure 2. Our robot arm configuration. (Top panel) The arm in its initial position with the second joint at 90 degrees. (Bottom panel) The arm oriented so that the pen is perpendicular to the front of the page. The red arrows show the direction of rotation for each joint.

A rotational joint has 1 degree of freedom, given by the joint's angle, which means our robot had 3 degrees of freedom, or 3 ways the robot could move in space. The number of degrees of freedom defines the minimum number of real numbers required to represent the robot configuration, so we had to know the angles of the three joints to determine the position of the pen. For more information on this topic, please visit the Modern Robotics lesson.

The aforementioned arm configuration and workspace transformation were the default design defined by the professor in the code. We tested this design with our control scheme using the arena shown in Figure 3 and found that we could move the pen close to the points we specified on the paper. Therefore, we chose this arm configuration and workspace transformation for our final design going into P-day.

We found additional characteristics of this design that we believed would help our robot draw a square on the paper. We observed that the robot arm did not experience much gravity sag in its initial configuration, which we believe was because the first motor was in the z configuration, so its axis of rotation was in line with gravity (-z direction). Also, the second and third joints had initial angles of 90 and 0 degrees, respectively, so this helped minimize gravity sag as the force of gravity extended through the length of the second and third arm segments. The arm base was placed high enough that it seemed to decrease the likelihood that the second or third arm segment would rise above the base, which would create a large amount of torque on the motor.

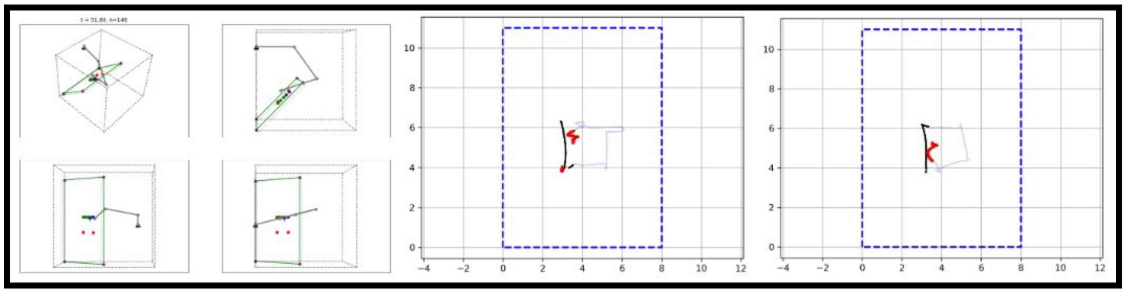


Figure 3. Example runs on the test arena. The paper and square specifications are labeled as "Initial test" in ./src/myarmsim.py. The green dots show the path the robot arm is trying to move the pen along, and the magenta dots at the corners are for calibration.

Finally, we wanted our robot to draw straight lines to reproduce the square, and we observed during testing that the arm could draw fairly straight lines (Figure 3). We believe that this could be attributed to the joint orientations in tandem with the control scheme. While the first joint rotates about the z axis, the next two joints both rotate about the y axis, so they can adjust the depth of the end effector with respect to the paper and compensate for the arc created by the first arm segment.

1.2.2 Control Scheme

Our software can be found in the ./src directory and a detailed explanation on our computational approach can be found in our ./howto/2020-P2-cyan-howto.pdf.

1.2.2.1 Autonomous mode

To activate autonomous mode, in which the robot autonomously moved the pen to any one of the four corners, the user had to press one of the following keys: **'w'**, **'e'**, **'r'**, or **'t'** (./src/myarmsim.py). Each key represented a different corner, and this key press called the move plan (./src/move.py) to move the robot. Within the move plan, we first used forward kinematics to determine the pen tip position (x,y,z coordinates). We obtained the corresponding initial motor angles from the function get_goal (from motorsim.py), and then plugged in those angles to the function getTool (from arm.py), which was called on an instance of the arm class that did not inherit any unideal motor properties. The function getTool returned the pen tip location by applying a 4 by 4 rigid transformation for the last arm segment on a 4 by 1 matrix representing the pen in homogeneous coordinates using the following dot product:

$$pen\ position \ = \ M_{4\times4} \bullet pen_{4\times1} \qquad (1)$$

Next, we set one of the corners of the square as the goal position we wanted to move to. We divided a straight path from our current position to this goal position into 5 equally spaced steps using numpy's linspace function. We thought that breaking down the path into smaller steps would help the robot stay on track and reduce its chances of moving to a position far from the goal position. We also thought it would help the robot draw a straight line, because if you break down a straight path into many small steps, even if the robot is drawing arcs, the combination of all those drawings would more closely resemble a straight line.

We then used the python library tinyik, for our inverse kinematics, which we used to determine the motor angles required to set the pen in the desired position. Using tinyik, we created another arm model with the same configuration defined in armSpec that we used to set its end effector position at each step and estimate the corresponding joint angles. We then fed these angles into the function set_pos to move the robot motors. After each move was completed, we set the previous goal position as our starting position and repeated the process until we looped around all the corners back to the starting corner.

To solve inverse kinematics, the library tinyik uses the BFGS algorithm, which is a solver method of the function scipy.optimize.minimize that attempts to minimize an objective function. The BFGS algorithm is an iterative solver for unconstrained nonlinear optimization problems that provides an approximate solution to inverse kinematics. In general, inverse kinematics cannot be solved in a closed-form solution. There may be no solutions, one solution or multiple solutions for the angles. The approach described above is a numerical inverse kinematics method, which means that it does not require robot-specific mathematics to solve inverse kinematics and can accommodate arbitrary robot mechanisms and task spaces (Source: University of Illinois inverse kinematics tutorial). This worked well for our situation, as we could change our arm design, such as the joint orientations or number of arm segments, and still use the numerical approach to solve inverse kinematics. With numerical inverse kinematics, you first make an initial guess at a solution and then iteratively drive your guess towards a solution. Unlike analytical methods, which allows you to find all possible solutions, numerical methods will only give you one solution that is a locally optimal solution based on your initial guess (Source: Modern Robotics, Chapter 6: Inverse Kinematics of Open Chains). One drawback of using numerical methods is that the solution may converge to local minima rather than the global minima when minimizing error.

1.2.2.2 Calibration

Our calibration is explained in detail in ./howto/2020-P2-cyan-howto.pdf. The goal of calibration was to determine an angle offset to apply to the angle configuration we set the motors at during each step taken while moving to a different square corner. This is to account for errors between the estimated joint angles and actual joint angles required to move the pen to the desired position. Calibration relied upon the move plan stated above. We decided not to do calibration during P-day because we weren't able to get calibration to consistently work while testing on the arena in Figure 3, and we found that the process took too long.

2. Results

The visualized results from P-Day for every team can be found here. In the visualized results, the pen was registered as drawing on the paper when it was located within a region above the page, indicated by a magenta box as seen in the 3D visualized results. A summary of the results is shown in Table 1. F signifies a failed run, P signifies meeting the pass/fail requirements, and NA signifies the team did not attempt the arena.

	Arenas									
Team Name	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Cyan	F	F	F	F	F	F	F	F	F	P
Firebrick	F	F	[1]	F	F	F	F	F	F	F
Indigo	F	F	F	F	F	F	F	F	F	F
Lime	F	F	[2]	F	F	F	F	F	NA	NA
Maroon	F	F	F	P	F	F	NA	NA	NA	NA
Peachpuff	F	F	P*	F	F	F	F	F	NA	NA
Razzmatazz	F	F	F	F	F	[3]	F	F	F	F
Tomato	F	F	P	F	F	F	F	F	F	F

Table 1: P-Day results for each team and their arena trials.

[1] Depending on whether the shape is considered closed, they may have passed.
[2] Part of the drawing extends outside the bounds of the paper, so it is unclear whether they passed.
[3] Depending on whether the shape is considered closed, they may have passed.
*Peachpuff arena 2 was the only pass where there was no paper puncturing, however, the shape drawn was not centered at the correct position nor was it overlapping the square given.

In Figure 4, the 2D visualized results are shown for the trials that met the pass fail criteria. Cyan's blue square (top left panel) is half the size compared to the other team's squares shape because Cyan used the scale to represent the full side length of the square. Therefore, Cyan's shape has side lengths of about 1 unit.

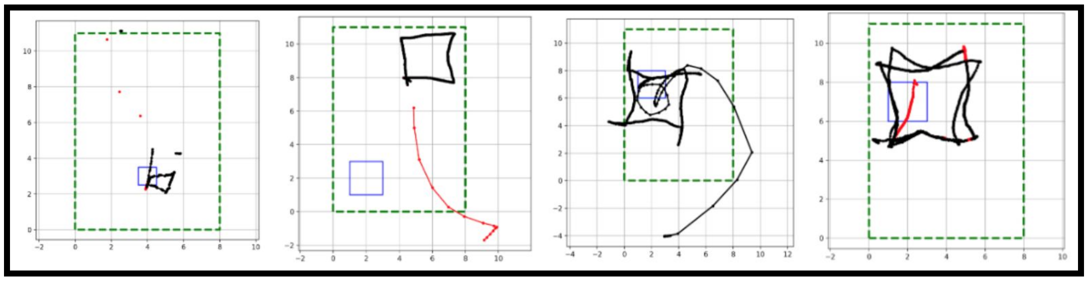


Figure 4. 2D results that met pass/fail criteria. (Far left) Cyan Arena 9, (middle right) Maroon Arena 3, (middle right) Peachpuff Arena 2, (far right) Tomato Arena 2.

3. Discussion

We only met the pass/fail criteria on Arena 9. A video (titled as "Chapter 2") of our run for Arena 9 that corresponds to this result can be found here. We believe that we were able to meet pass fail criteria on Arena 9 for the following reasons: (1) at the end of each move to a square corner, we set the previous goal position to the starting position for the next move. The arm had a tendency to miss the goal position, so this correction helped bring the arm back to the path connecting the previous and next corners and draw a closed shape. (2) The arm and workspace configurations seemed to work well for Arena 9. As shown in Figure 5, the arm had to bend inward on itself to reach the square. This positioning, while awkward looking, may have been favorable as the arm segments were closer to the base and the robot moved the second arm segment underneath the first one, so there was less torque from gravity on the arm segments. It should be noted that although the pass fail requirements require pushing into the paper from the front side, the results were the same if the robot approached the paper from the back side, so long as the pen tip was in between the dotted green and dotted magenta boundaries as seen in Figure 5. (3) Our robot arm could draw straight lines. This was the case for Arena 9, and in Figure 6, we show two additional examples on different arenas in which the robot could draw straight lines. This supported what we observed during testing. As stated in our design section, we believe the arm configuration and small steps helped the robot draw straight lines.

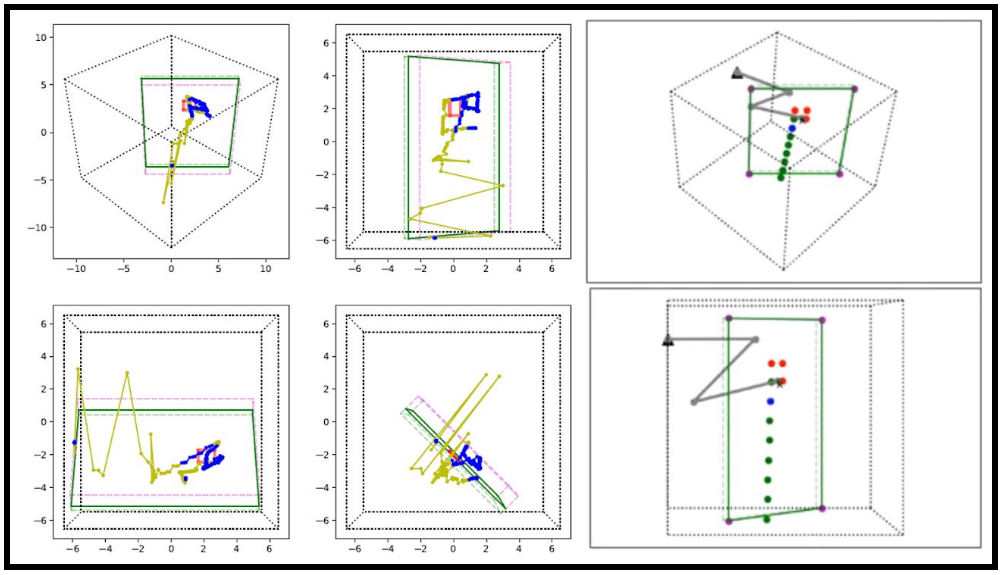


Figure 5. Cyan team's Arena 9 trial that met pass/fail criteria. (Left frame) 3D visualized results. (Right frame) Different orthographic views of the arena with robot arm (gray lines), move trajectory (green dots), and square corners (red dots) visualized.

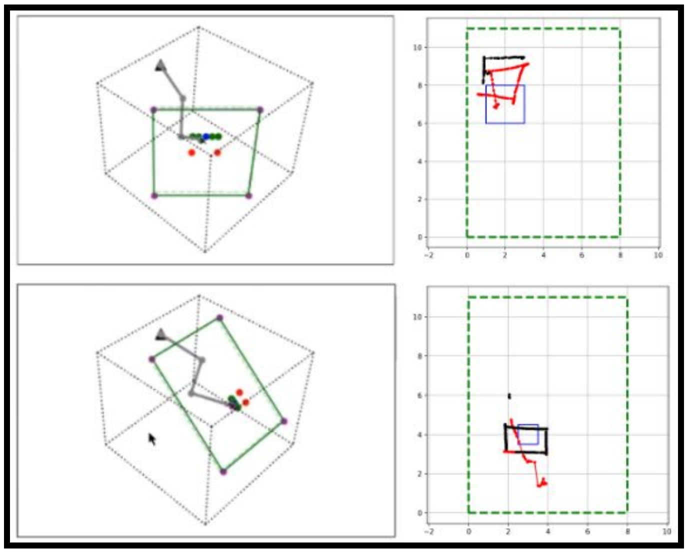


Figure 6. Two examples of our robot drawing straight lines. (Top panel) Arena 2 (video: titled "Chapter 3") with 2D drawing. (Bottom panel) Arena 5 (video: titled "Chapter 3") with 2D drawing.

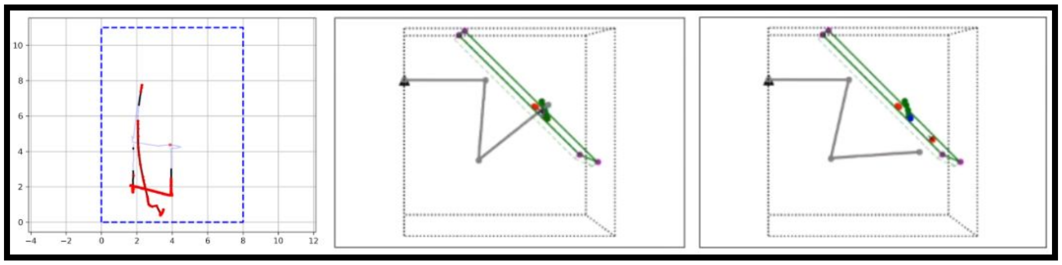


Figure 7. (Left) Example drawing that shows our robot's inability to maintain consistent depth while drawing the square for Arena 5 (video: titled "Chapter 7"). (Center) Robot arm is too "shallow", drawing would register as light blue lines as seen on the left. (Right) Robot arm is too "deep", drawing would register as red lines as seen on the left. The arm also overshoots the goal position, possibly due to gravity sag.

There is much room for improvement in our robot design. Our robot was inconsistent in its performance across the different arenas, and we believe this happened for multiple reasons. First, as previously stated, the arm had a tendency to miss the goal position with various magnitudes of error. It would sometimes head in the right direction, then overshoot at the end, or miss the target altogether and veer off drastically, which sometimes resulted in the motors erroring (motor error means they shutdown and cannot be moved). This could be because the inverse kinematic solver gives an estimate for the joint angles without taking into account unideal motor properties and gravity. Therefore, there is an error between the estimated joint angles and the actual joint angles required to set the pen at each goal position. Calibration was supposed to account for these errors, but as mentioned in our arm design section, our calibration process took too long to complete, so we did not use it on P-day.

Second, the arm and workspace configurations were less favorable for certain arenas than others. For example, on Arena 5, the arm could draw straight lines and make a closed shape, but it could not maintain the correct depth within the page, as shown in Figure 7. It appeared that the arm experienced gravity sag as it moved in the -z direction to draw one of the sides and moved further down than it should have. The arm had to extend further from its base compared to its configuration in Arena 9, which would create greater amounts of torque. For other arenas, for example Arenas 3 and 7, the robot was unable to draw something that resembled a closed shape. It is unclear why this happened, and more testing should be conducted to help reveal causal mechanisms.

Therefore, to improve our robot, we recommend adjusting the arm and workspace configurations by placing the workspace so that the arm is closer to the center of the arena and higher up, possibly even outside the workspace. This might enable the robot to reach more of the squares without experiencing motor errors. This should be validated with testing, and one way to test could be to iterate through all the possible angle configurations and plot the end effector positions using forward kinematics, as Maroon team did and explained in their howto. In addition, the torque values, along with motor temperature and error codes (from motorsim.py) should be calculated for each end effector position to see whether those positions can be reached without causing the motors to shut down.

We also recommend trying an alternative inverse kinematic solver, such as the one used by the Maroon team that was explained in their howto. Maroon team used the Levenberg-Marquardt algorithm implemented by Scipy's least_squares function. This is also an iterative algorithm, but this algorithm prioritizes a solution of angles that are closest to the robot arm's current joint angles. This would encourage the robot to take the steps that minimize changes in movement from the current configuration, which could help prevent the robot from moving into configurations that may cause high torques and motor errors. We observed that in some trial runs, the robot raised its second arm segment above its first arm segment to reach the desired point, rather than bend its arm segment under the first arm segment, which appeared to require less change in configuration and would have resulted in lower torques.

Furthermore, we recommend that the calibration process be made faster so it takes less time to use, whether that be during a test run for debugging purposes or during a P-day trial. The reason why our calibration process took so long was because the user had to manually adjust the pen position of the robot after moving to each calibration point to calculate an error. Maroon team developed a control scheme that allowed them to manually move the robot arm based on the pen location in cartesian coordinates, which we believe is a more intuitive way of controlling the robot and is one thing that we could try implementing to make the process of manual adjustment during calibration faster.

Ultimately, we should have done more testing on as many different arenas as possible before P-day. We did most of our testing on one arena, which was not used during P-day, and we were able to generate the different P-day arenas prior to P-day using the script genP2.py. Testing on more arenas would have better informed our arm and workspace design. We also could have done calibration and saved our calibration settings for each arena before P-day, so that on P-day we could just load in the calibration settings for the given arena. This way, we would not have to rely on calibrating right before our P-day run and be pressed for time.