MEAM 5200 Final Project Report

Shrehari Prsad Gopalakrishnan, Raj Anadkat, Ryan Jacobowitz, and Brandon Reid

December 27, 2022

1 Objective

Robotic arms are a commonly used tool in industrial processes to accomplish tasks quickly and efficiently. The goal of this project was to implement algorithms learned in the class to stack blocks as high as possible while facing off against other teams. A 7 degree of freedom PANDA arm was used with 7 revolute joints. There were two types of blocks, four static blocks on a stationary table, and twelve dynamic blocks on a rotating platform. The dynamic blocks were shared between both teams. To score points, these blocks needed to be placed on a scoring platform which was next to the static block platform. Scoring for each block was based on the formula $Points = Value \cdot Altitude$ where value was 10 for static blocks and 20 for dynamic blocks and altitude was the distance of the center of the block from the scoring table. Two teams would then face off and the team with the highest point total at the end of a 3 minute time period would win. In the case of a tie, the winner would be determined first by the number of dynamic blocks stacked and finally by the time it took to stack the blocks.

After every team was able to play two matches, all the teams that were able to win enough times would then move to the knockout stage of the competition, where a tournament style competition would occur. In this stage if both teams were able to score 4000 points or more, the competition would continue for 2 more minutes, allowing teams to stack more blocks and see the full capabilities of each team's implementation. In addition, there were two possible robot configurations, red and blue, which the robot needed to account for, as each team would be randomly placed on one of the robots during the competition.

2 Methodology

2.1 Strategy

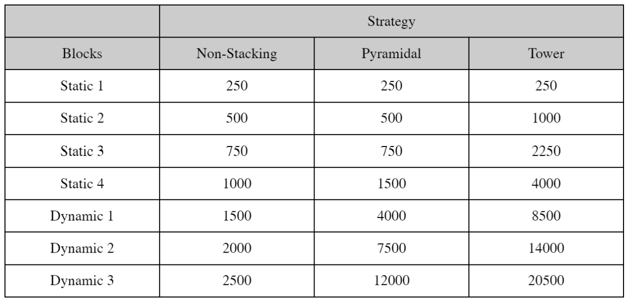
Given the goal of scoring the most points possible, the following requirements were set for the project in the following order of priority:

1. To have a strategy that would maximize the amount of points per block
2. To be able to grab static blocks
3. To be able to place blocks on the scoring platform
4. To be able to stack blocks on the scoring platform
5. To meet timing objectives
6. To be able to grab dynamic blocks

If these requirements were met, the robot would be able to effectively score points and hopefully win the competition. The priority was chosen because it was most important to come up with an effective strategy, as the rest of the project would flow from this strategy and the rest of the requirements were created based on this strategy. Being able to grab static blocks was the next step, as in order to score any points at all, blocks would need to be grabbed by the robot, and the static blocks were the easiest blocks to pick up. The success of this grabbing process was determined by how reliably the robot was able to pick up static blocks as a percentage. The placing was the next step, as to score any points at all, blocks would need to be placed on the scoring platform. Similarly, the success of this step was determined by the percentage of times a block was placed on the platform over a number of trials. After getting blocks grabbed and placed properly, a stacking algorithm was created to get more points per block.

The success of stacking was determined by the number of blocks stacked and the reliability in stacking a target number of blocks. Next the robot was required to meet the 3 minute time limit, and so after determining the robot could successfully grab, place, and stack blocks, each of the algorithms created would be optimized to reach a time of 30 seconds per block. Lastly, the dynamic block grabbing was measured by the total number grabbed and the reliability of grabbing the dynamic blocks as a percentage. This was the lowest priority requirement, as even though dynamic blocks would give a significant amount of points, this process was the most difficult part of the project. Placing of the dynamic blocks was determined to be the same as the static blocks, and so was not a separate requirement. In addition to these requirements it was also important to keep the robot within joint limits, not go out of bounds (under the table) and keep the movement of the robot safe.

To find a successful strategy, the points amounts were calculated for a variety of strategies to find which would provide the best results. First the non-stacking case was considered, where blocks were not stacked at all and were instead placed on the platform without stacking. This case was beneficial because if it was determined that stacking was too difficult or took too much time, this algorithm could focus instead on the dynamic blocks. The next strategy was to create a pyramidal structure, which would have three static blocks as a base for stability and then stack a tower on these blocks as high as possible. This strategy has the benefit of making a stable tower and if there were problems in stacking a tall tower without tipping over, this pyramid would be a good alternative. The last strategy considered was the tower strategy which would stack blocks in a single column as high as possible.



	Strategy		
Blocks	Non-Stacking	Pyramidal	Tower
Static 1	250	250	250
Static 2	500	500	1000
Static 3	750	750	2250
Static 4	1000	1500	4000
Dynamic 1	1500	4000	8500
Dynamic 2	2000	7500	14000
Dynamic 3	2500	12000	20500

Figure 2.1: Point Totals for a Variety of Strategies

The results of the point calculations can be seen in figure 2.1. The strategy with the highest points per block ratio was the tower method, as stacking blocks as high as possible gave significantly more points than getting more blocks. From these calculations it was immediately apparent that the non-stacking strategy was not sufficient, since it required grabbing a significant amount of blocks for limited rewards. The pyramidal structure did score well, but while testing it was determined that stacking was not the main issue the robot was facing, and the tower was fairly stable given a good enough stacking algorithm. It was determined that it was significantly more difficult to grab dynamic blocks, and since the pyramidal structure requires a lot more blocks to be point efficient, the tower was determined to be the best method. Other issues with the pyramid are that placing would need to be both more accurate for the first layer of blocks, and blocks would get in each other's way on this first layer, since these blocks would be placed so close together, the robot had difficulty fitting the end effector in such a way to place the blocks properly.

2.2 Implementation

2.2.1 Inverse Kinematics

From this strategy, the other requirements of the robot were derived and the first step was to grab static blocks. To accomplish this task, first analytical inverse kinematics was attempted. This inverse kinematics algorithm would be given an end position and rotation for the robot to go to and output multiple possible solutions for the robot's joint angles. These solutions would be filtered to only show the solutions that meet joint limit requirements. This implementation would provide the first major challenge for the project. The solution would output correct results for certain positions, but would not work for other solutions, and these inconsistencies eventually lead to using a gradient descent based inverse kinematics result. While it would take more time to solve for a solution, this gradient descent based solution would produce results more accurately and more consistently than the analytical solution. Gradient descent based inverse kinematics was achieved by using the following algorithm:

1. The inputs are the desired end effector target position and the seed value for the gradient descent.
2. Set the initial q to the seed.
3. Find the change in q by solving the difference between the target and current q transformation matrices. Solve for the difference in translation and skew in the rotational change.
4. Find the change in q with respect to the center of the joint limits.
5. Solve for the Jacobian of q.
6. Find the projected version of the centered change in q by multiplying each joint by the Jacobian and normalizing this result.
7. Find the actual change in q by solving dq by combining this centered projection with the change needed to reach the target calculated earlier.
8. Update q based on this dq value.
9. Repeat steps 3-8 until the maximum number of steps is exceeded or the minimum step size is greater than the norm of all the changes in q.

2.2.2 Static Block Grabbing

Upon completing this inverse kinematics algorithm, a grabbing algorithm was implemented. The robot would start in the given starting position, and then would move to a joint position above the static blocks. This position was determined by hard coding in the joint angles for a position that could see every block on the static platform. Hard coding of joint angles allowed for less computational time, and since the robot would constantly come back to this position, it was set as a global variable called "detect position". The robot would then detect the blocks using the camera, choose the closest block to the end effector and compute the inverse kinematics of the position of this block. The seed value for inverse kinematics was chosen as the detect position. Next, the robot moved to the joint angles found by the inverse kinematics in two steps. First it set each joint angle to the necessary joint angles except for joint 2, and then the robot rotated joint 2 to pick up the block. This two step process was done because if the robot moved directly to the block position, the robot would sometimes collide with the top of the block as it was rotating and moving downward. Delaying the movement of joint 2 allowed the robot to rotate to the correct position above the block, and then to move downward and pick up the block with no collisions. Upon reaching the correct position, the robot closed its gripper and grabbed the block.

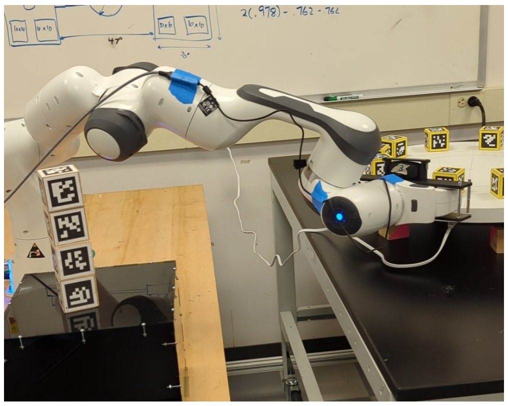


Figure 2.2: Dynamic Position Before Sweep

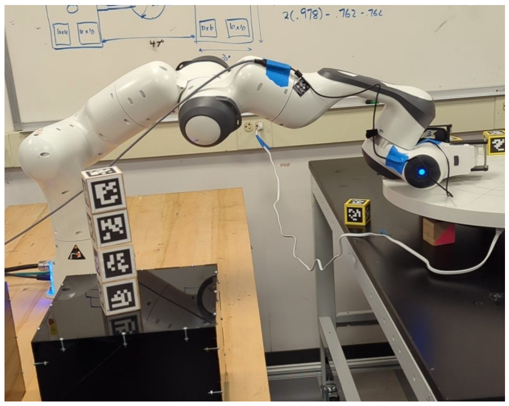


Figure 2.3: Dynamic Position After Sweep

2.2.3 Static Block Placing/Stacking

After creating a grabbing algorithm, the next step in the process was to place and stack blocks on the scoring platform. This placing initially used the same inverse kinematics algorithm used in the grabbing algorithm. For a block in the first position, this algorithm did solve for the correct position, but when going into a stacking algorithm (placing at a higher position), the robot would fail to reach a local minimum far more often. To solve this issue, first many different seeds were tried to see if there was a seed that would provide the correct results. Unfortunately, even trying seeds that were practically next to where the robot needed to go failed to solve for the correct position. To rectify this, the robot was hard coded to specific joint angles during the placing part of the program. This provided a consistent result for the placing and also had the added benefit of reducing computation time, since no inverse kinematics was solved during this process. To find these positions, an iterative process of finding the best placing joint angles for each subsequent block was done until the proper amount of blocks were stacked. This process could go as high as the robot was capable of going, as long as it was pre-programmed in. To go from grabbing to placing, the robot would first go back to the detect position and then to a new "place position" that was directly above the scoring platform. These two positions were used to provide for a smooth, safe, and consistent movement from platform to platform without worrying about knocking other blocks. As the stack grew taller, the place position was always higher than the highest block, so there were no issues with knocking over the stack during this section. After reaching the place position, the robot would move to the joint positions pre programmed to the current stack height and upon reaching this new position, the robot would release the block. This process of grabbing and stacking was repeated until every static block was stacked.

2.2.4 Dynamic Block Grabbing

The robot was first programmed to grab dynamic blocks by reaching a pre-set `dynamic_position` and waiting for the block to be detected by the camera. This method, however, was found to be too slow as the robot took too long to calculate the inverse kinematics and the dynamic block had often moved by the time the robot tried to grab it. In an attempt to fix this issue, the team added different rotations to the robot in order to try to catch up to where the block had moved. However, it was eventually determined that it would not be possible to accurately calculate the future position of the dynamic block in time for the competition. As a result, the team tried a different approach: using the robot to sweep over the blocks and try to grab one after it had passed over the table. This method proved to be surprisingly effective, as it consistently allowed the robot to pick up dynamic blocks during lab tests. As an added advantage, it would also knock some blocks off the table leaving fewer blocks for the opposite team to stack. It was important to ensure that the robot was not too close to the rotating table and that it moved in the opposite direction of the table's rotation, so that the blocks were moving into the gripper. Figure 2.2 and 2.3 demonstrate the sweeping motion of the robot.

2.2.5 Pseudocode for Entire Program

The program used the algorithm below to grab and stack both dynamic and static blocks. Tables 2.1 and 2.2 has each of the hard coded values for each variable in the code.

1. Move to the given starting position
2. Determine whether the team is red or blue and set `detect_position`, `place_position`, stacking position variables, and dynamic position variables depending on which color the robot is.
3. Move to `detect_position`
4. Use `detect_blocks` function to get the pose of every static block with respect to the camera.
5. Find the closest block to the end effector based on an L2 distance.
6. Using the `align_gripper_block_rotation` function, the gripper was aligned with the block such that the X and Y axes of the block and the end effector were aligned.
7. Using the equation `H_new` = `H_current` @ `H_ee_camera` @ `H_block`, the necessary joint angles were found to grab the block.
8. Using `H_new` as the target pose for the end effector, inverse kinematics was done to find the joint angles for the desired target position.
9. Using the `safe_move_to_position` command (using the two stepped method explained in 2.2.2), the manipulator approached the block, and grabbed it using the `exec_gripper_cmd`.
10. Once the block was grabbed, arm moved back to `detect_position` and then to `place_position` ensuring that it does not hit the other static blocks or the current stack of blocks on the scoring platform.
11. The robot moved from this position to hard coded stacking joint angles depending on how high the current stack is.
12. The robot opened the gripper and moved back to `place_position`.
13. This process of grabbing and stacking (steps 3-12) was repeated until every static block was stacked.
14. Moved to dynamic positions going from one position to another to ensure a pre-planned smooth transition ending just above the table, before the blocks with a horizontal gripper.
15. Sweep around the table such that the dynamic block slides within the gripper of the robot by moving to 3 intermediate positions.
16. Sleep for 10 seconds to let blocks rotate into gripper.
17. Close the gripper and proceed to stacking blocks as outlined in steps 11 and 12.

3 Evaluation

In each step of the implementation testing was done in simulation to see how each step of the implementation met the requirements set out for it. After these tests were done in simulation, results were found in lab time to make sure the theoretical results matched the actual performance. If the requirements were not met, further changes to the implementation would need to be made. The final iteration of the gradient descent based inverse kinematics had two main issues. First, a seed was required to be used as an initial guess for the algorithm to do gradient descent on. Many times the chosen seed would reach a local minimum before reaching the solution, and for some positions, the algorithm failed to get any solutions regardless of how many seeds were tried. This only occurred when placing the blocks in the first position, as the algorithm

Variable Name	**Blue Team Values**
Start Position	[-0.01779206, -0.76012354, 0.01978261, -2.34205014, 0.02984053, 1.54119353+$\pi/2$, 0.75344866]
Neutral Position	[0, 0, 0, -$\pi/2$, 0, $\pi/2$, $\pi/4$]
Detect Position	[$\pi/12$, 0, 0, -$\pi/2$, 0, $\pi/2$, $\pi/4$]
Place Position	[$-\pi/12$, 0, 0, -$\pi/2$, 0, $\pi/2$, $\pi/4$]
Q Place 1	[-$\pi/12$, $14\pi/128$, 0, -$76\pi/128$, 0, $86\pi/128$, $\pi/4$]
Q place 2	[-$\pi/12$, $10.5\pi/128$, 0, -$76\pi/128$, 0, $84.5\pi/128$, $\pi/4$]
Q place 3	[-$\pi/12$, $7\pi/128$, 0, -$76\pi/128$, 0, $84\pi/128$, $\pi/4$]
Q place 4	[-$\pi/12$, $3.5\pi/128$, 0, -$76\pi/128$, 0, $84\pi/128$, $\pi/4$]
Q place 5	[-$\pi/12$, $2.5\pi/128$, 0, -$72\pi/128$, 0, $79\pi/128$, $\pi/4$]
Q place 6	[-$\pi/12$, $5\pi/128$, 0, -$63\pi/128$, 0, $70\pi/128$, $\pi/4$]
Dynamic Position 1	[0.75-pi, 1.07043798, 0.52569093, -1.14401406, pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]
Dynamic Position 2	[0.75-pi, 1.07043798, 0.52569093, -1.14401406, pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]
Dynamic Position 3	[1.2-pi, 1.07043798, 0.52569093, -1.14401406,pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]

Table 2.1: Blue Variable Amounts

Variable Name	**Red Team Values**
Start Position	[-0.01779206, -0.76012354, 0.01978261, -2.34205014, 0.02984053, 1.54119353+$\pi/2$, 0.75344866]
Neutral Position	[0, 0, 0, -π / 2, 0, $\pi/2$, $\pi/4$]
Detect Position	[-$\pi/12$, 0, 0, -π / 2, 0, $\pi/2$, $\pi/4$]
Place Position	[$\pi/12$, 0, 0, -π / 2, 0, $\pi/2$, $\pi/4$]
Q Place 1	[$\pi/12$, $14\pi/128$, 0, -$76\pi/128$, 0, $86\pi/128$, $\pi/4$]
Q place 2	[$\pi/12$, $10.5\pi/128$, 0, -$76\pi/128$, 0, $84.5\pi/128$, $\pi/4$]
Q place 3	[$\pi/12$, $7\pi/128$, 0, -$76\pi/128$, 0, $84\pi/128$, $\pi/4$]
Q place 4	[$\pi/12$, $3.5\pi/128$, 0, -$76\pi/128$, 0, $84\pi/128$, $\pi/4$]
Q place 5	[$\pi/12$, $2.5\pi/128$, 0, -$72\pi/128$, 0, $79\pi/128$, $\pi/4$]
Q place 6	[$\pi/12$, $5\pi/128$, 0, -$63\pi/128$, 0, $70\pi/128$, $\pi/4$]
Dynamic Position 1	[0.75, 1.07043798, 0.52569093, -1.14401406, pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]
Dynamic Position 2	[0.75, 1.07043798, 0.52569093, -1.14401406, pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]
Dynamic Position 3	[1.2, 1.07043798, 0.52569093, -1.14401406,pi/2-0.35, 1.30507887+pi/4, -1.30833-0.05]

Table 2.2: Red Variable Amounts

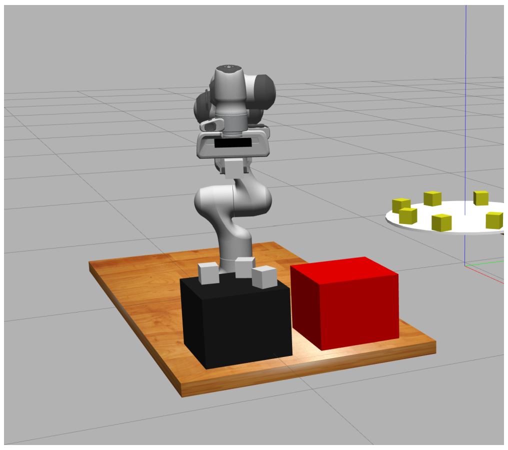


Figure 3.1: Successful Block Grab in Simulation

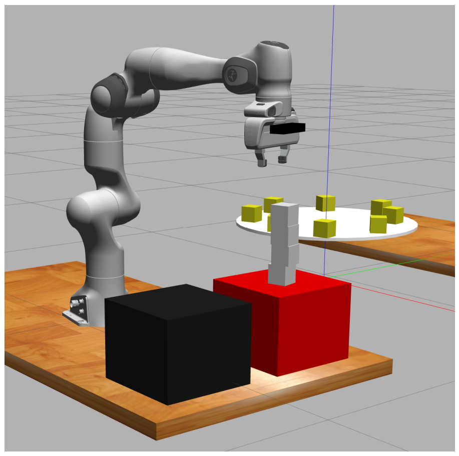


Figure 3.2: Successful 4 Block Stack in Simulation

worked reliably when grabbing blocks or when placing the first block of the stack. To solve this issue, the placing positions were hard coded into the program before hand to provide for consistent results. This would have the added benefit of reducing computation time as the inverse kinematics did not need to be solved during this step. The second main issue was that the algorithm was relatively slow, sometimes taking 10 seconds just to solve for the correct solution. To solve this problem, the minimum step size and maximum number of steps were changed to allow for convergence faster.

After implementing the grabbing of static blocks using this inverse kinematics algorithm, the program was tested by running various configurations of blocks in simulation and in lab hours to determine how consistent the algorithm was. After running these tests in simulation ten times, it had a 100% success rate during initial testing. While these results were satisfactory initially, during further testing it was determined that there were occasionally times when the algorithm would output a result that would exceed joint limits. This error was because joint 7 could have multiple angles that satisfied the inverse kinematics which were a difference of π from one another. In order to correct this the joint would need to be subtracted by $\pm\pi$ if the joint exceeded the joint limits in either direction. Unfortunately this error was inconsistent and did not occur in simulation very often, and so was not fixed in time for the competition. After the competition, using simulation and lab results, the fixed code provided a 100% success rate in grabbing and picking up a block after 10 trials of the grabbing program. The placing algorithm had a similar success measurement, and was incredibly consistent. After finding the correct joint angles, every trial was a success, with a 100% success rate after 10 trials of placing and stacking given a block was grabbed. The grabbing and stacking was also tested together and after 10 trials also provided a result of 100% success. These 10 trials were done in simulation, but the robot was also tested in hardware, consistently being able to stack blocks. There was no instance where a stack fell over from a faulty stacking algorithm.

The program was not as fast as required initially, as the time it took to compute the inverse kinematics was 10 seconds for each block, resulting in a stacking speed of 45 seconds per block, resulting in the robot barely being able to stack all four blocks within the 3 minute time period. This was not ideal, as any slight delays in calculation could result in not making a full static block stack, and since any ties were broken by the amount of time it took to stack blocks, slow calculations were a major concern. To fix this, the minimum step size and maximum number of steps were changed. Lowering the maximum number of steps and increasing the minimum step size was risky, as there was a chance the solution would not find the exact solution within a decreased number of steps. It was therefore a matter of balancing accuracy with speed, and the optimal result was determined to be a maximum number of steps of 200 and a minimum step size of 0.001 after iterating through a number of different configurations. Some of these iterations, the time it took to calculate the inverse kinematics for each iteration, and whether or not the results were consistently accurate can be seen in Table 3.1. The unit of time in this table is the time it took the simulation to run,

Max Number of Steps	**Min Step Size**	**Time to Calculate**	**Accurate**
300	1e-4	22.40	True
200	1e-4	15.23	True
300	1e-3	19.90	True
200	1e-3	11.88	True
100	1e-3	7.38	False
200	1e-2	0.67	False

Table 3.1: Inverse Kinematics Timing

which was greater than the number of seconds it took to run on the actual robot. The worst result took approximately 10 seconds to run on the actual robot while the 200/0.003 case took approximately 5 seconds to run, which is consistent with the magnitude difference found in simulation. Using this faster approach on the physical hardware during a lab section, the static blocks were able to be stacked in just under 2 minutes, which meets the 30 second per block requirement.

The dynamic block stacking was not implemented properly until after the competition, but was tested using a percent reliability of successful grabbing. After implementing the sweeping algorithm properly, the program was able to grab dynamic blocks in 5 out of 6 of the test cases in lab, and was able to successfully stack these blocks each time. This reliability was incredibly high and allowed for a consistent dynamic block stacking algorithm. Other important measurements to consider were whether the tower was knocked over during any point in the stacking process, and whether the robot was safe in its movement. With the final code, the robot never knocked into the tower during any testing or simulation, so it was determined that it was reliable in not knocking the tower over. In terms of safety, there were multiple measurements to make, such as whether another block was hit while holding a block, whether the robot stayed within joint limits, whether it refrained from hitting either the static or revolving tables, or whether it made any movements that were too fast or erratic. The final implementation was successful in passing all of these measurements by not knocking into the tables or blocks, not making erratic movements, and by remaining within joint limits consistently for every test in simulation and hardware.

4 Analysis

During the competition, the robot was successful in stacking 3-4 block within the 3 minute time period. Figure 4.1 shows a successful stack of four blocks made during the competition. The strategy chosen was incredibly successful, as there were many teams that were unable to grab a single block because they tried to go for dynamic blocks initially. By going for static blocks initially, the robot was much more consistent than teams that went for dynamic blocks first and was able to successfully make multiple stacks of four. During one of the rounds of the competition, another team tried to go for dynamic blocks initially, failed, and then went for the static blocks, which caused them to stack the static blocks more slowly. This caused our robot to win because of stacking our blocks before they could stack their blocks. This strategy of going for the static blocks first, while simple, was an incredibly effective method for doing well during the competition, and was a resounding success. Because the robot was able to make consistent stacks, our robot was able to make it to fourth place during the competition, even though there were many robots that had more sophisticated dynamic block algorithms.

There were many issue faced by the robot during the competition. The first main issue was the timing of the inverse kinematics. During the initial stages of the competition, the robot took a long time to calculate the inverse kinematics. It took approximately 10 seconds, and caused the robot to miss the fourth block on the stack during one of the rounds, and barely place the fourth block in another round. The changes to the step size and maximum number of steps were made halfway through the competition, between the qualification and knockout rounds. During the knockout rounds, however, a new issue was faced by the robot in the form of joint limit 7. Joint 7 consistently converged to a solution outside of the joint limits, and so the robot stopped after the third block in the final two rounds of the knockout phase. Regardless of these errors, the robot was still successful during much of the competition, reaching fourth place overall. This was a major accomplishment, as the dynamic portion of the code had errors and it was not possible for the robot to pick up a dynamic block. Successes in strategy and in the stacking algorithm were what allowed for the robot to reach as far as it did.

Multiple augmentations were made to the code after the competition. First, the timing of the robot was perfected by iterating the inverse kinematics for different values of the parameters. Next, the possible solutions for joint 7 were changed so that the solution would never exceed joint limits. The final and most difficult augmentation was to the dynamic portion of the code. The dynamic joint positions were determined through iterating through positions until the desired sweeping motion was achieved. It required sufficient distance from the table to prevent collision with the table while still being able to be low enough to grab a block in its center. The robot was eventually able to do the desired motion and was able to grab a dynamic block fairly consistently (5/6 trials during lab). The dynamic algorithm was tested during a lab section as can be seen in figures 2.2 and 2.3, and the robot was eventually able to stack 6 blocks as can be seen in figure 4.2. This was only tested on the physical hardware once, but was still a major success.

The process of creating this robot involved a lot of challenges and lessons learned along the way. The inverse kinematics was difficult to get working properly, but working through how minimum step size and maximum number of steps changed the time of running gave us a much better understanding of the logic behind choosing the correct parameters for a gradient descent based algorithm. Even if the inverse kinematics was solving properly, it was crucial to the requirements that timing requirements were met, and it was both a challenge when trying to figure out how to accomplish this, and a reward when adjusting the parameters achieved the desired result. It was also interesting to see how a very simple algorithm could work better than more complicated robots, as our robot was very consistent in placing the blocks properly even when just using hard coded values. It is important to prioritize the most important section of a project given that every project has a limited time frame, and it was a good learning experience to look at the problem and decide what is most important. Trying to implement a perfect dynamic algorithm or a stacking algorithm using inverse kinematics would have taken time away from the grabbing and stacking of static blocks. If we had focused on doing something more complicated, the static block section of our program would have suffered, and we would not have done as well as we did. Learning how to operate the robot safely and within the limits given by the competition was also an important learning experience. When working in simulation, it can behave far differently from real life. No simulation is perfect, and figuring out how the simulated and lab/competition results compared was an important thing to learn. An example of this would be the timing of the robot, as the simulation was far slower than the actual robot, so it was difficult to gauge how quickly the robot would move in real life. Making sure to plan for these differences is an important lesson to take away from this project.

While the robot had some errors during the competition, the competition code was able to do surprisingly well given the limited abilities of the dynamic block algorithm. The final code is a overall a success, as it meets every requirement. It is capable of reliably grabbing, placing, and stacking the static blocks into a stack of 4000 points. It is capable of reliably grabbing the dynamic blocks and stacking them on top to create a stack of 5-6 blocks. It is capable of meeting its timing constraints, as it is able to stack five blocks in under 3 minutes and six blocks in just over 3 minutes. It is able to maximize the score total per block by implementing a strategy that focuses on building tall, and by going for static blocks first, it was able to consistently score 4000 points efficiently without relying on the success of the dynamic block algorithm. It is also safe, as it successfully avoids the stacked tower, the tables, and makes safe movements that are not too quick or dangerous. By successfully meeting every requirement and implementing many algorithms learned in the course, this robot was successfully in its objective of teaching us about how 7 degrees of freedom robots work and learning the challenges and rewards from working with real hardware.

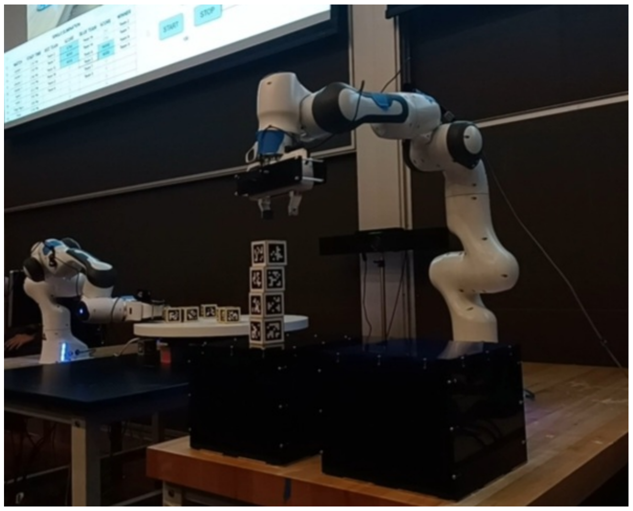


Figure 4.1: Competition Stack Results

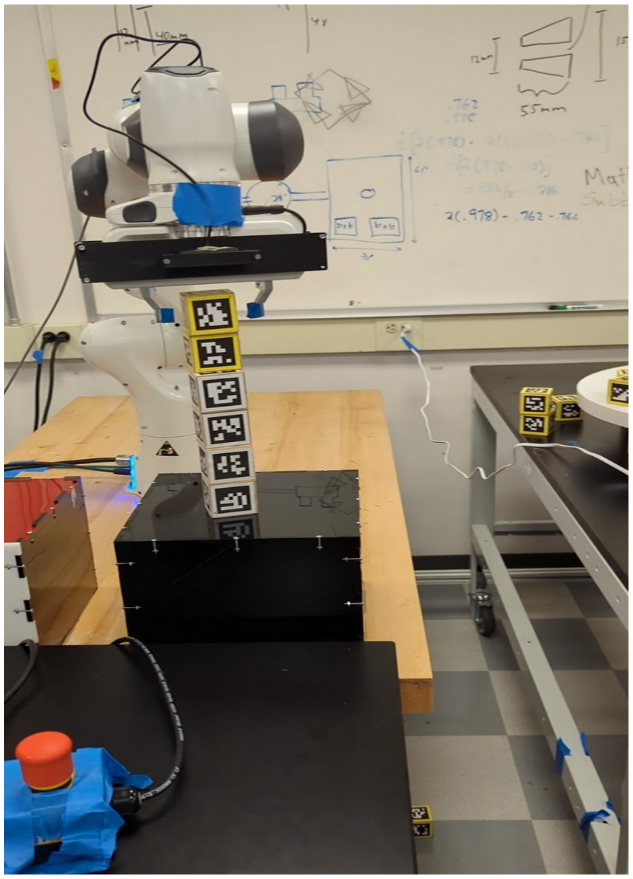


Figure 4.2: Best Stack Results