Final Project

Jinwei Bi, Rubo Xing, Richard Chen, Jianxu Chai

1 Method

For this part, we will introduce our methods about static blocks and dynamic blocks. For the static blocks part, we will show how we pile the four blocks, and how we flip the block to make it white face upwards. Similarly, for the dynamic blocks part, we will show our basic process as well as our implementation.

1.1 Static blocks

We will introduce our basic process of grabbing dynamic blocks first and give the details of how we did that. Then, we show our implementation and how we deal with the bonus part (flipping the block to make it white face upwards).

1.1.1 Basic process

To grab the static blocks, we decided to open the gripper first. Then, we move the end effector to the top of the first static block (about 0.1m above the block). Next, we move the end effector down to the center of the block and close the gripper to grab the block. After that, we lift up the end effector and move it to the target platform and leave the block. Then, we move the end effector to grab another block. One thing to nice is that we need to move up the end effector 0.05m at the previous target position to pile the four blocks one by one.

1.1.2 Homogeneous transformation matrix

To move static blocks, we first need to move the end effector to the position of the static block. Therefore, we first need to get the location of the static blocks. We know that we are provided with the homogeneous transformation matrix relating the frames of the blocks to the frame of the camera (T_{block}^{camera}), which are Tag1 to Tag12. Also, we know the homogeneous transformation matrix relating the frames of the Tag0 to the frame of the camera (T_{Tag0}^{camera}) is provided in Tag0. Since we want to get the homogeneous transformation matrix that helps us locate the blocks with respect to the robot's frame (T_{block}^{robot}), we need to calculate it using the equation below:

$$T_{block}^{robot} = T_{Tag0}^{robot} T_{camera}^{Tag0} T_{block}^{camera} \tag{1}$$

To use that equation, we found we have already got T_{block}^{camera}. To get T_{camera}^{Tag0}, we just need to calculate the inverse of the Tag0, according to the equation below:

$$T_{camera}^{Tag0} = (T_{Tag0}^{camera})^{-1} = (Tag0)^{-1} \tag{2}$$

To get the T_{Tag0}^{robot}, we can calculate by hand according to the information on the instruction. The result of it is showed below:

$$T_{Tag0}^{robot} = \begin{bmatrix} 1 & 0 & 0 & -0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

Because Tag0 is fixed, we calculate the homogeneous transformation matrix relating the frame of the camera to the frame of the robot at the beginning to save time. The result of it is showed below:

$$T_{camera}^{robot} = T_{Tag0}^{robot} T_{camera}^{Tag0} = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

Thus, to get the homogeneous transformation matrix relating the frame of blocks with respect to the robot's frame, we only need to calculate it using the equation below:

$$T_{block}^{robot} = T_{camera}^{robot} T_{block}^{camera} \tag{5}$$

1.1.3 Inverse Kinematics

After we get the homogeneous transformation matrix relating the frame of blocks with respect to the robot's frame, we can use inverse kinematics method to move the robot's end effector to the block. We use "IK.inverse()" function from Lab 3 to move the end effector to the target position. A quick review of that function is that it uses gradient descent to solve the full inverse kinematics of the Panda robot. It takes two inputs: one is "target", which is the desired 4x4 transformation matrix from the end effector to world; The other one is "seed", which is a 1x7 vector of joint angles[q0,q1,q2,q3,q4,q5,q6], which is the "initial guess" from which to proceed with optimization. The output of the "IK.inverse()" function are q, success, rollout. The "q" a 1x7 vector of joint angles giving the solution if success if True or the closet guess if success is False. The "success" is True if the IK found a configuration which achieves the target within the given tolerance. Otherwise it's False. The "rollout" is a list containing the guess for q at each iteration of the algorithm.

To make the IK work, we need to find a good seed. Therefore, we adjusted the joint configurations many times until we found a good seed around the stationary platform and another good seed around the goal platform where we place the blocks.

For the blue team, in practice, we have mistakenly used the seed near the goal platform for the all the process of moving static blocks. Fortunately, the IK function could find the configurations which achieve the target during the whole process. The seed we used for the static blocks part of the blue team is [0.08,0.05,-0.25,-2,0,2*π/3,0.3]. The position of that seed is showed in Figure 1 below.

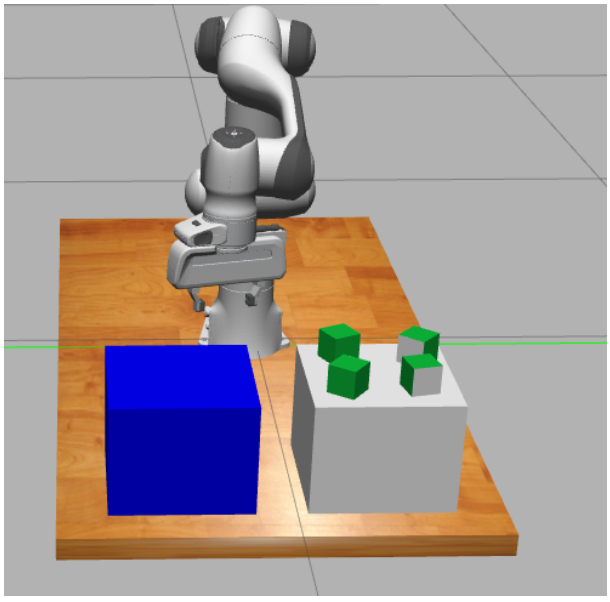
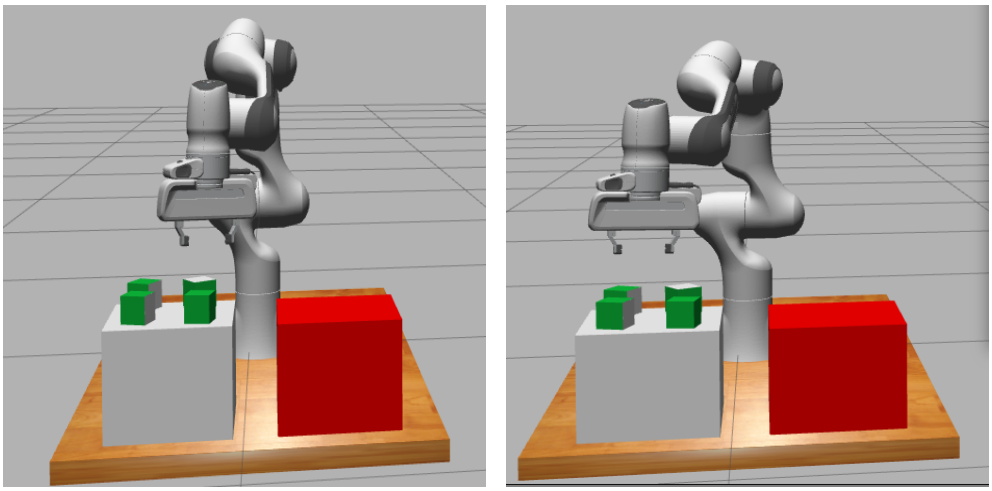


Figure 1: The position of the seed used for static blocks part of the blue team

For the red team, in practice, we also mistakenly used two seeds not the same as the seeds we used at the beginning. But the IK could also generate the configurations to successfully move the end effector to the target position. The seed used for picking up the blocks is [0.08,0.05,-0.25,-2,0,2*π/3,0.3], while the seed used for placing the blocks is [-0.08,0.05,-0.25,-2,0,2*π/3,0.3]. The position of that two seeds is showed in Figure 2 below.



(a) Seed for picking up blocks (b) Seed for placing blocks

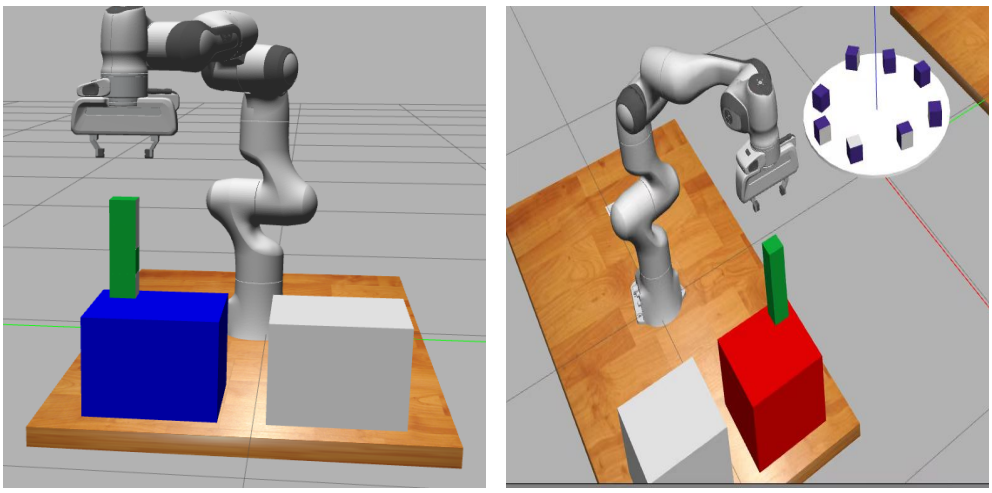
Figure 2: Two seeds used for static blocks part of the red team

1.1.4 Implementation

For our implementation, we first need to find static blocks, which are tag1 to tag6. Then, we calculate the homogeneous transformation matrix relating the frame of the block to the frame of the base of the robot according to the Equation 5. Then, we need to multiply the rotation matrix of the T_{block}^{robot} by -1, since we want the end effector faces downwards instead of upwards. Next, we add 0.05 to T_{block}^{robot}[2][3] because we want the end effector first be moved to the position 0.05m above the block, and use IK.inverse() function with the target of the inputs set as T_{block}^{robot} to get the joint configuration q. Then, we use command "arm.safe_move_to_position(q)" to move the end effector to the position that is 0.05m above the center of the upper face of the block.

After that, we subtract 0.075m from T_{block}^{robot}[2][3], and use IK function again with the target of the inputs set as T_{block}^{robot} to get the joint configuration q. Then, we use command "arm.safe_move_to_position(q)" to move the end effector to the position that is the height of the center of the block. After the end effector moves to the center of the block, we use the command "arm.exec_gripper_cmd(0.049)" to grab the block. This command means that the gripper closes to a width of 0.049 meters. Since the width of the block is 0.05 meters, we believe it can grab the block and it does in practice.

After grabbing the block, we first move it up 0.2m by adding 0.2 to T_{block}^{robot}[2][3], and use it as the input of IK function. After lifting it up, we move the end effector to the goal platform and place the block. Considering the dynamic blocks, we finally decided to place the static blocks at the upper right corner for red team and upper left corner for the blue team. The positions to place the static blocks are showed in Figure 3 below. One thing to notice is we should add the height of the target position after placing each block, since we want to pile the blocks. After reaching the target position, we use the command "arm.exec_gripper_cmd(0.09)" to open the gripper and let the block fall. After placing the block, we lift up the end effector by adding 0.1 to T_{block}^{robot}[2][3] before moving the end effector to grab another block. Since lifting it up can avoid collision with the blocks and the platform. Then we move the end effector to grab another block and repeat the whole process mentioned above until piling up the four static blocks at the goal platform.



(a) The position to place static blocks for the blue team

(b) The position to place static blocks for the red team

Figure 3: The positions to place the static blocks

1.1.5 Bonus (flipping the block to get white side up)

In this section, we brainstormed some ideas to flip the blocks and get white side facing up in one time in order to save some time during the competition. Initially, we decided to grab the block vertically and lay the end-effector horizontally when putting the block so that it can be turned over. But after some careful consideration, we think it could be too dangerous to use this method especially when picking, flipping and placing for the first block, since the wrist part near the end-effector has volume, the arm may collide with the target table when placing the block. For conservative reasons, we gave up this idea and thought of another way to flip the blocks in one time.

In stead of grabbing the blocks vertically, we decided to grab the block with the end-effector at a 45-degree angle to the vertical. Figure 4 shows an example of a specific orientation of the end-effector when grabbing the block with tag4 being detected. From the figure, the annotated x, y, z coordinate shows the orientation of the end-effector. The main idea here is to let the robot grab the boarder of the detected tag and tag6. When placing the block, we can just rotate the end-effector in 90 degrees with respect to its y axis. In this way, we can flip the block in just one time during the moving process.

Our goal is to determine homogeneous transformation matrix of the end-effector. To fulfill this, we need to figure out the transformation matrix of the end-effector in the block coordinate frame, R_{ee}^{block}. Here, we need to just consider the rotation part because the position of the end-effector can be easily determined by the transformation matrix of detected tag in the robot arm frame. Base on the previous discussion, we can express R_{ee}^{block} as:

$$R_{ee}^{block} = \begin{bmatrix} 0 & 1 & 0 \\ c(\pi/4) & 0 & -c(\pi/4) \\ -c(\pi/4) & 0 & -c(\pi/4) \end{bmatrix} \tag{6}$$

Then we can use the below equation to calculate the rotation part of the end-effector's transformation matrix:

$$R_{ee}^{robot} = R_{block}^{robot} R_{ee}^{block} \tag{7}$$

In the above process, we have classified situations according to the types of tags. We found that the above analysis is the general solution for tag1, tag2, tag3, and tag4, which can be used to achieve the bonus goal. For tag5, we have to implement the above method twice in order to make a 180 degrees rotation of the block. And for tag6, we can directly grab the blog vertically and placed it with no rotation or flip.

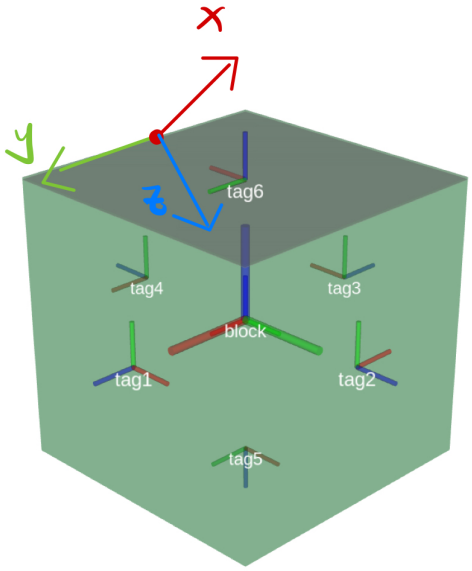


Figure 4: One example of how to grab and place the block when tag4 is detected (on the top side)

1.2 Dynamic blocks

1.2.1 Basic process

For the dynamic blocks, the hard part of grabbing them is locating where they are. There is no real-time feedback of where they are as the table rotates. So basically we just set the grab them in a passive way. We first set the gripper close to one side of the rotating table and set the gripper to 45 degrees to the XY plane. Then horizontally move the gripper onto the plate and let the blocks slide into the gripper and clamp. Lastly, we put the first block on the target table and repeat the process with a higher target position. Our final goal is to get two blocks. As time goes by, the location of the dynamic blocks on the rotating disk would be quite different from the original state, which makes it harder.

1.2.2 Implementation

Based on the process we have for static block, we use the same technique on dynamic part including the homogeneous transformation matrix and inverse kinematics. The difference is the seed for the inverse kinematics in this part is adjusted to the configuration where the end effector is close to the rotational table. The seed used for the goal platform remains the same.To grab the blocks, we first set the seed configuration near the side of the rotational table [2*π/5,0.4,-0.25,-1.6,0,2*π/3,0.7], then horizontally move onto the edge of the table by changing the Y position of T_dynamic_target. For avoiding the collision of the gripper with the table while still having the function of stopping the rotating blocks, we rotate the gripper 45 degrees using the rotation matrix R:

$$R = \begin{bmatrix} c(\pi/4) & 0 & s(\pi/4) \\ 0 & 1 & 0 \\ -s(\pi/4) & 0 & c(\pi/4) \end{bmatrix} \tag{8}$$

And then set the waiting time there for 9 seconds. The waiting time is long enough for letting one cube move into the gripper. Then we close the gripper exec_gripper_cmd(0.045), and let it move to the target using the results from IK.

2 Evaluation

Basically, we divided our final project into three small sections, one for static part, one for dynamic part, and another for flipping side part. For each section, we both implemented code and evaluate the performance of the code on lots of tests (simulation and real-life). After making sure that all these parts worked well, we then combined them together based on our strategy or plan (i.e. grab two dynamic blocks first and then stack all static blocks on two of them).

2.1 Metrics

In general, we designed our tests based on the following metrics:

IK success

Our code implementation are largely based on the inverse kinematic algorithm. And for this algorithm, we found that choosing a proper seed is really important. In the outputs of IK.inverse(), there is an important value to indicate whether a valid solution is found or not. So, we printed out the this output for each run of the IK function.

Joint limits warning or collision detection

During the simulation and tests, we sometimes encountered warning and error messages. To eliminate those errors, we just literally stopped our tests immediately and modified the code (may related to the seed choice) to make sure that our solution is conservative (i.e all warning and error messages disappeared). In addition we need to also make sure that no point on the robot (except the base) is ever went below an invisible horizontal planar barrier with an altitude of .200m in the world frame.

Successful result (strictly)

The final success of the result was not enough. During the simulation and tests process, we focused our attention on the movement of the arm. We need to foresee if there are any potential collisions once the setup of the environment change. We need to also make sure that the end-effector moved to a position that was what we expected for each movement. If something went wrong, we just wrote it down and fine tuned them.

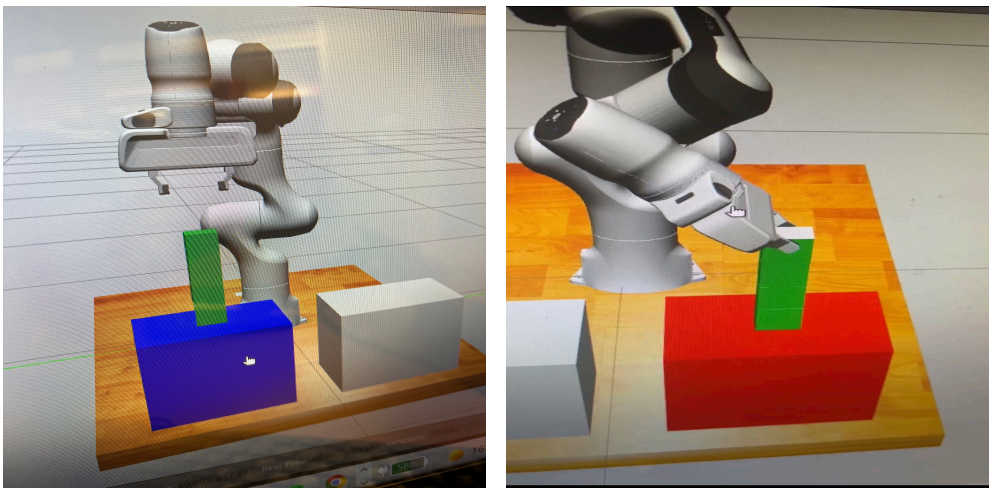
Time evaluation and strategy optimization

Different strategies can have different results in terms of the score getting and time spending. We need to optimize the trade-off between this and anticipate other situations. For example, our initial plan was to grab two dynamic blocks first in case that we were sharing 8 dynamic blocks and place them aside. Then, we wanted to stack the static blocks and finally put the dynamic blocks on the top. However, after test in real-life, we found that this strategy exceeded the time limits (3 minutes). So, we had to optimize it to save some time or think of another strategy. More detailed about strategy are given in the analysis section below.

Based on the above metrics, the results of our tests are shown as follow:

2.2 Static Part

For the static section, as we have limited physical lab time, we paid more attention to the simulation tests in Gazebo environment. This was good because every time we reset the test, the environment changed. Therefore, we can try lots of scenarios to ensure that our solution worked well. We also tested our solution in the physical lab Figure 6.



(a) The result of simulation for blue team (static part)

(b) The result of simulation for red team (static part)

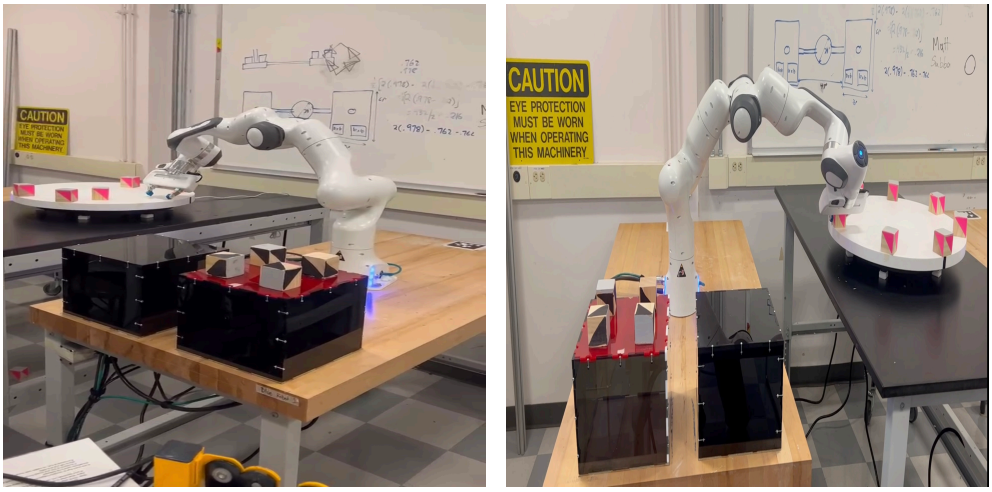
Figure 5: The result of simulation (static part)

Figure 6: The result of physical test (static part)

2.3 Dynamic Part

For the dynamic section, as we can not use the position information of the dynamic blocks, we paid more attention to the physical tests in real-life Lab environment. We fine adjusted the position of the end-effector during the grabbing process to grab the blocks successfully and avoid any hard collision between the robots and the blocks. It is worth noting that here we set a wait time in our code to let the robot wait for the blocks to come into the middle of the gripper. Compared to another strategy that is to detect whether the gripper grasped a block using the function $arm.get\_gripper\_state()$, the waiting strategy can ensure that we can grasp the near center position of the block for each grasp if we set enough waiting time. However, the drawback is that we may waste some time if the block is already in the middle of the grippers. Another one is that if we did not set a long enough time, we may not grab a dynamic blocks, which may influence on the blocks stacking afterwards. Interestingly, we eyeballed the speed of the turnable table and found that it takes the table around 15 seconds to spin 90 degrees. And to consider

the extreme case, we need only to set the waiting time to be about half of that (8-9 seconds). Note that, it is hard to test in simulation. So, we only show results from real test here (Figure 7).



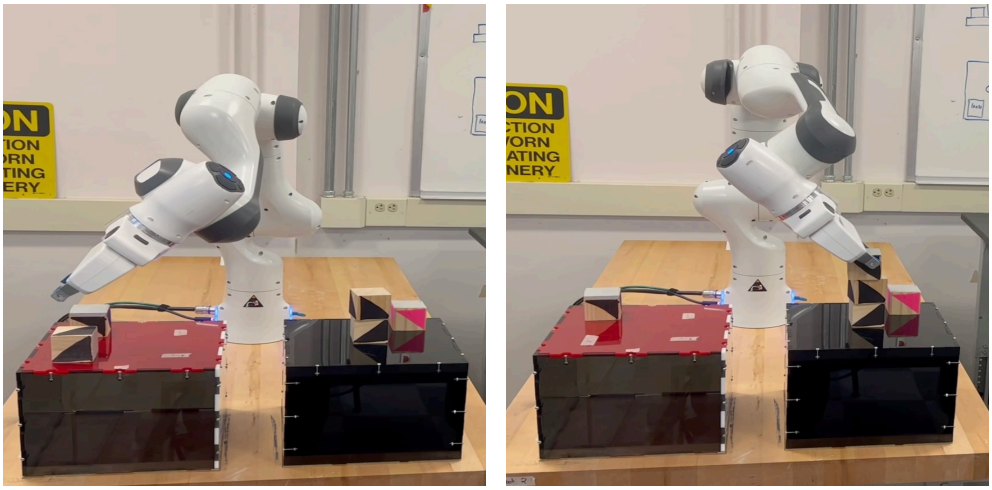
(a) The results of real tests for blue team (dynamic part)

(b) The results of real tests for red team (dynamic part)

Figure 7: The results of real tests (dynamic part)

2.4 Flipping Side Part

In this section, we paid attention mainly on both of the simulation tests. We found that our solution worked for some case (Figure 5(b)) but not for some particular cases (when the blocks are on the far side and the white side is pointing to the opposite direction of the robot) shown in Figure 10. Here is one successful result in real life.



(a) Before grabbing

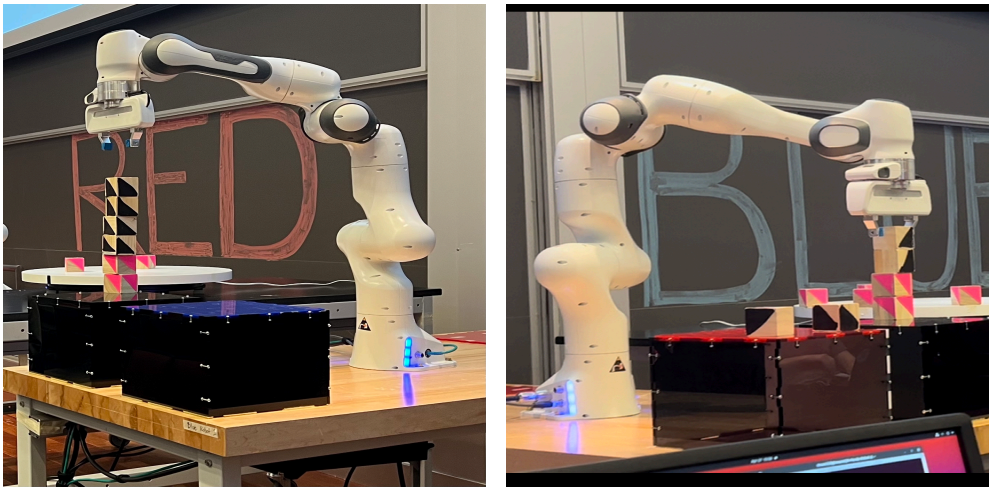
(b) After placing

Figure 8: The result of physical test (flipping side part)

2.5 Final demo

As team 2, we did a good job in our first round as a blue team (9(a)). And we successfully entered the quarter-finals based on that. However, in the second match as a red team, we did not grasp two dynamic blocks successfully due to our insufficient setting of waiting time (7 seconds). So, we updated the waiting time to be 9 seconds. And we also updated our code for the red team trying to stack 7 blocks (4 statics

+ 3 dynamics) before quarterfinals. But in the quarter-finals, we lost for some reason (gripper seemed not working well) as a blue team. Therefore, we felt like it was a pity that we have not been able to see whether we can successfully stack 7 blocks. Overall, our hard work has paid off.



(a) final demo blue team (b) final demo red team

Figure 9: final demo

3 Analysis

For this part, we will analyze our strategy and its performance for simulation. We will also analyze the robot's performance during the lab and the final demo.

3.1 Strategy and simulation

We believe our final strategy is a great solution to this project. The reason why we tried to first grab dynamic blocks is that we know that other teams may try to grab dynamic blocks at the beginning. If we grab them later, not only the number of dynamic blocks will be smaller, but also the location of the remaining dynamic blocks will be destroyed by our opponent. Therefore, we decided to grab two dynamic blocks at the beginning and then grab the static blocks. After calculating the time of grabbing static blocks, we realized we only left about 80 seconds to finish grabbing dynamic blocks. According to the time we left for dynamic parts, we finally decided to grab two dynamic blocks.

We also discussed whether we flip the blocks to make the white face upwards or not. We tried to achieve flipping blocks at the beginning of our project. However, we then found that it is a difficult, time-consuming and unstable task, especially when turning the block with "tag5", which need to be flipper twice to make the white face upwards. Also, sometimes the solution of the IK function may contradict with the joints' limits. Like the block with the red circle in Figure 10, when we tried to grab that block with 45 degrees and turn the white face upwards, we encountered the joints' limits when trying to grab that block. Therefore, we finally decided to give it up and tried to spend that time grabbing dynamic blocks, because we noticed that flipping the block with white face upwards only give us 50 points bonus, however grabbing one dynamic block can at least give us 2x(25+50)=150 (all the dynamic blocks have white face upwards) more points.

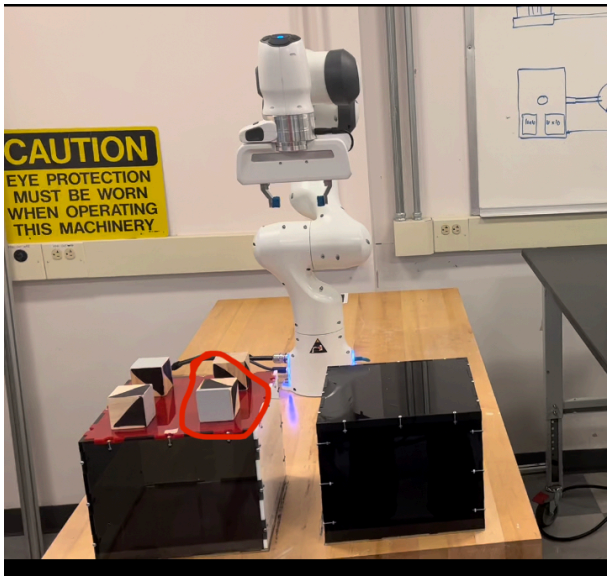


Figure 10: Configuration that we encountered joints' limits when trying to flip the static block

We then discussed about where we place the blocks and the sequence of the blocks. At first, we considered grabbing two dynamic blocks and putting them at the corner of the goal platform at the beginning and then piling the four static blocks at the center of the goal platform. After that, we put those two dynamic blocks on the top of the four static blocks. Because we know that the higher the dynamic block it is, the much more points we will get. We are proud that we have achieved that during the robot lab (Figure 11), however, that strategy costs us three and a half minutes. Therefore, we had to change our strategy.

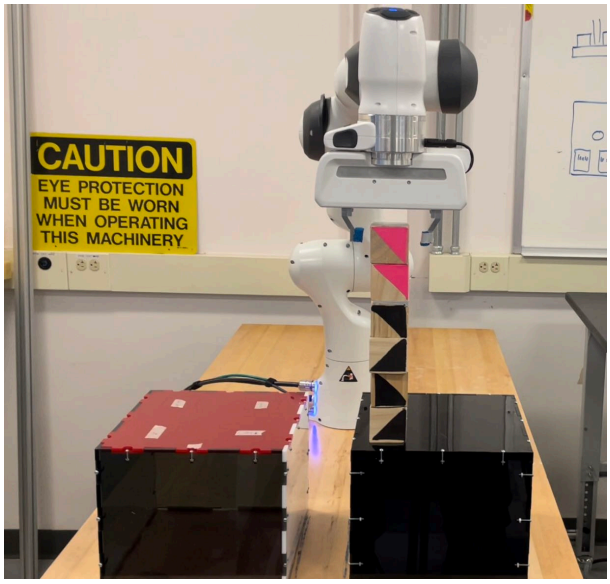


Figure 11: Result at the robot lab: two dynamic blocks on the top while four static blocks at the bottom

Finally, we decided to put two dynamic blocks at the bottom at the corner of the goal platform first and then put four static blocks on top of them, which only costs us about 2 minutes and 40 seconds. The reason why we put those blocks at the corner of the goal platform, upper right for the red team and upper left for the blue team, is that position is at the middle position between the initial dynamic blocks and static blocks.

Although we can't simulate grabbing dynamic parts during simulation, we believe it has basically

achieved what we expected in simulation.

During the simulation part, we have successfully moved the end effector to the target position of dynamic parts, as well as piling four static parts.

3.2 Results in practice

Although the static parts work well during the simulation, we have encountered a lot of issues during the robot lab. First of all, we assumed ”lag0” at the first of the list, while in practice, ”lag0” can be anywhere of the list. Therefore, we changed our code to make it find ”lag0” at the beginning. Also, we found that we have to tune up the parameters that lifting up the block after grabbing it, since in the robot lab, our initial parameter that works well for simulation may be smaller in practice, since we encountered collisions after lifting the block up.

In addition, we actually spent more time adjusting dynamic parts during the lab. In practice, we noticed that the start time plays a really important role in dynamic parts. If we pressed the enter key to initiate the robot right after a block passed 9 O’clock position, the robot could grab two dynamic blocks successfully. Otherwise, the robot may collide with the dynamic blocks. The reason is that we set the waiting time for the end effector at the edge of the rotating plate is 7 seconds for our first code. Therefore, if the start time is out of the suitable window, the robot may unfortunately grab only one dynamic block or nothing for the dynamic part. That’s why we tried to adjust the waiting time to 9 seconds during the second round competition at final demo. If we know the speed of the rotation of the turn-able plate and the real-time location of the dynamic blocks, we may grab the blocks more accurately with less time.

Also, we noticed that the seed for dynamic parts also need to be adjusted for each team. In practice, We found good seeds for red team, however, when we tried to use those seeds for blue team, it failed. We found the robot arm would twist for a circle before reaching to the dynamic blocks. Therefore, we tuned the seeds until the robot could smoothly move to grab the dynamic blocks.

4 Lessons Learned

In many complex systems, especially those with moving parts or those that are open to the outside world, a variety of unplanned events can occur. Visual feedback would allow a robot to react to changes, like tracking and grabbing a moving block rather than throwing a Hail Mary on the hope that the block will move into the claw. The ability to adapt to changing conditions greatly expands the realm of possibilities.

Gradient descent is not guaranteed to find the global minimum of a function, so to make our IK solver work well, we needed to set it up with a good seed. This is similar of a wide variety of situations where total optimization is infeasible and highlights the need for good heuristics. In this case, the robot arm was manually positioned so that it was close to the blocks to constrain the configuration space and decrease the probability of becoming stuck in a local minima.

Simulations are wonderful ways to test and debug, but by definition, are not a substitute for the real deal. Our simulations were not perfect. The dynamic table did not spin and the physics engine is simplified from the actual laws of physics. There are varying levels of simulation depending on whether one wants a high level overview of the systems, adding in the software, to even adding in some hardware. Ultimately, simulation is an important step but it alone is not enough.

Importance of planning for goals in advance. Some of the most important steps take place before writing the code and debugging. The fastest computer in the world would not make a difference if it had to run bogosort. Likewise, poor algorithm choice or planning can ground the operation before it even takes off.

5 Acknowledgement

We would like to thank Professor Nadia as well as the Teaching Assistants for your help and suggestions.