MEAM 520: Final Project Report

Group Partner Names:
Jayadev Chevireddi, Luis Escobar, Alice Li, Sahachar Tippana

Due Date: December 14 2021

1 Goal

Our primary goal for this final project was to generate safe, reliable, collision-free paths for a Franka Panda arm to pick up static and dynamic blocks and place them on a goal platform, such that stack height was maximized and blocks were placed white face up. Our secondary goal was to maximize points within the given time frame by triaging between (1) tasks that were time consuming yet are deterministic and are sure to provide more points (e.g. manipulating blocks to be oriented white face up) and (2) tasks that are more probabilistic but could provide high reward in a short time only if executed well (e.g. picking up dynamic blocks).

2 Methods

Before we began programming, our team went ahead and brainstormed an overall plan and necessary functions we would need to tackle this challenge. This plan is summarized in Figure 3. It had to be adapted multiple times due to changes in the information provided by the TAs, as well as adjustments due to errors, or the correctness of our approach. Despite these changes, the main ideas and functions brainstormed were kept throughout the development of our project.

We divided our project into five sub-problems, which could be solved separately. The first dealt with determining the necessary transformations we had to apply to the block poses, to get poses from the camera frame to the robot base frame. The second, and most basic task, involved finding safe poses for the robot to move to in its configuration space and approach working zones. The next task was to develop an algorithm to pick and place the static blocks without reorientation. Then, we had to develop an algorithm to reorient the blocks in the manipulation space. The final sub-problem was to develop an algorithm to fish for some some dynamic blocks. The methodologies for the last sub-task was altered various times since there was a change in the real-time updates we would receive in the competition, where we would essentially fly blind.

2.1 Transforming Poses into Robot Frame

We used the relationships between the transformation matrices shown in the top right corner of Figure 1 to determine the block poses in the robot base frame, frame 2, denoted in the diagram.

One important remark that needs to be made is how we orient the end-effector to pick up the blocks. Since at the start of the competition, we are given orientation data of each block, to use this information to generate the orientation of our end-effector, we took the target block orientation and multiplied it by a pure rotation around the `x-axis` by 180 degrees. In doing so, we guarantee that our end effector is aligned or super-imposed with the block and we can pick it up in a reliable manner.

2.2 Searching for Safe Poses

The next problem we tackled was programming an algorithm to approach our end effector to safe positions over the static blocks, over the target table, and an approach position for the turntable. Our general approach here involved using `calculateFK`, to calculate forward kinematics end effector homogeneous transformation

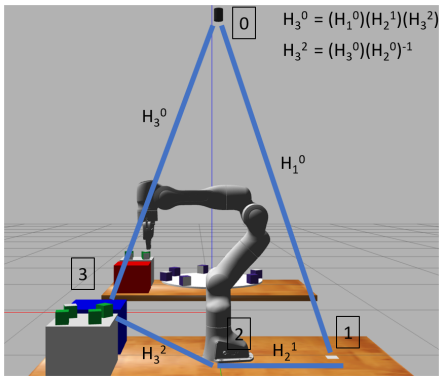

Figure 1: The required matrix multiplications to transform the block tags in the camera frame to the robot base frame.

matrices (from lab 1) and `solveIK` to calculate poses using this given matrix using inverse kinematics (from lab 3). We decided not to use a trajectory planner here, since there were no obstacles in our trajectory, and there were no important constraints for the route that our end-effector took.

One challenge with using the inverse kinematic function, `solveIK`, was using good seeds for the solver. A good seed was necessary for the solver to successfully determine a pose for the robot, given a homogeneous transformation matrix for the end effector. To tackle this, we took two approaches. The first involved creating an intermediate pose by directly tweaking joint angles. This was done in order to pre-position our robot in a configuration closer to the desired end pose. It also meant that we would not need to call `solveIK` to find a pose which deviated significantly from its seed. The main joints that we manipulate directly were q1 to get an intermediate pose almost over our targets (target block, platform, turntable), and q6 to position our arm with the desired pose to pick up and deposit the blocks. These changes were done directly from the home position of the robot.

The second approach we used to guarantee seeds and small movements in the space, was the use of a short function that divided the changes of the pose into small differential changes. Even though this meant feeding smaller incremental movements into `arm.safe_move_to_position()`, it made us more confident that we could use `solveIK` more reliably. In order to minimize the movement of the robot, we do not use all the intermediate poses that we pre-calculated. We used only a succinct set of them, which appeared sufficient for safe movement.

2.3 Picking up Static Blocks

The next part of our methods was to pick and place static blocks with no reorientation. The main purpose of this part of the algorithm was to work with the simulation extensively before any testing with the real robot. It also allowed us to guarantee the stacking capability, and ensure some points for the competition in case the other functions were not finished on time. The steps we used to pick and place static blocks were:

1. Start from a previous pose (safe position hover over the static blocks table).
2. Move the robot to a given distance over the given target static block “P1”.
3. Move down with the open gripper.
4. Close the gripper.
5. Move back to the “P1”.
6. Calculate intermediate poses to deposit the block.
7. Move to the target position with a given dz “P2”.
8. Move down to the table.

9. Open the gripper.
10. Move back to "P2".
11. Go back to 1.

2.4 Prioritizing Blocks

To pick up the blocks in an effective way, with the three minute time limit for the competition, we used a simple strategy. The method to reach the blocks was the same as in the case of picking the blocks with no orientation as discussed above. In terms of prioritization however, the blocks that had the white face on the top face were picked first. The second priority was given to the faces having the white faces that were neither on the top nor on the bottom face, while the cases where the white face was in the bottom face of the block were considered to be the last in the priority list. This order was chosen because the first case did not require the robot arm to perform any re-orientation of the block. This movement required the least amount of time to be stacked. The second case took only one rotation operation, while the third one required two. Once this priority order was defined, the robot would position its end effector to align with the homogeneous transformation matrices of the ordered blocks.

```
order = ['tag6','tag1','tag2','tag3','tag4','tag5']
```

2.5 Re-orienting Static Blocks

For re-orienting static blocks, we used the following approach. First, 'tag6' required no orientation, since it already had its white face up. 'tag1' to 'tag4' required 1 rotation of the block to get their white face up. While 'tag5' required two. We deduced that having 'tag5' performing two rotations is time consuming. Instead, having 'tag5' act as 'tag6' and reaching out to the dynamic blocks had potential to save time for dynamic points and hopefully rack up more points. Hence, we performed no rotations for 'tag5' and instead considered it similar to 'tag6' and simply used its homogeneous transformation as the desired end-effector pose. For the tags from 1 to 4, the grabbing of the blocks is similar to that of in the other two cases but the placing of the blocks is different. To face these blocks white face up, the desired homogeneous transformation matrix was determined by rotating the tags' original homogeneous transformation matrix by $-90°$ along the y-axis.

More elaborately, the reorientation of blocks can be divided into two stages. The 1st stage of reorienting of blocks involves modulating the pickup in such a way that for block with tags 1 to 4 would always have their white face to the same adjacent side when stacked without the 2nd stage of reorientation and making sure that the gripper does not hold the white face while pick-up. This means that the `move_to()` function has been modified to rotate the target-effector pose with $90°$ about the z-axis, making sure that the blocks with tags 1-4 are always pick-up in a certain fashion such that the white face is always determined to be the same adjacent side.

The 2nd stage of reorientation involves playing with the `place_block()` function and the stacking sequence code in the `main()` function to stack the blocks with all the white faces up. In the `place_block()` function, the desired end-effector pose is rotated by $-90°$ about the y-axis such that the end-effector reaches the tower position with the white face up at a safety distance (height) of 15cm in z-axis direction above the tower. From this intermediate pose, the joint configuration to place the block is determined by computing the inverse kinematic solution for the end-effector pose translated in the z-direction by negative of the safety distance.

For Tag-5, although not reoriented for the competition, the function `place_block()` rotates the block twice to obtain the white face up. This is achieved by performing another pick-up action (with the desired end effector pose rotated back by $+90°$ about y-axis) after the 2nd stage of reorientation of the tag5 block on the tower. Once, picked up from the tower, the 2nd stage of reorientation is repeated by rotating the desired end-effector pose by $-90°$ along the y-axis to reach the tower and release the block with white face up. The tag5 reorientation has not been used for the final competition for strategic reasons concerning the time limit to stack as many blocks as possible. However, the tag5 reorientation has been successfully implemented and can be observed from the figure 2.

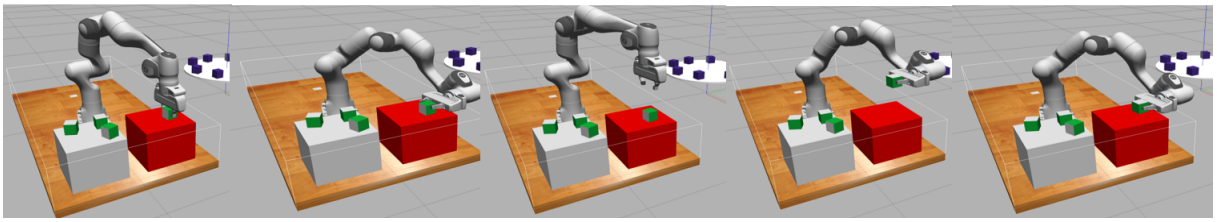

Figure 2: Sequence of operations for successful reorientation of the block with tag5.

2.6 Picking up Dynamic Blocks

To pick up dynamically moving objects, we opted for different strategies for the competition vs for simulation since the information provided for both settings were sufficiently different. For the competition, we were provided initial positions of blocks, but after that, would be flying blind. Given that we were not provided tag updates, certain assumptions had to be made about the block positions and orientations to pick up blocks: the blocks had not moved from its initial starting position to the time of pick up; the blocks were placed perfectly so that orientations were exactly as shown from the pose information for each of the given tags; and that the angular velocity of the table remained constant throughout its motion.

In terms of strategies for picking up dynamic blocks in the competition, a hover, pulse, and grasp mechanism was used. The hover position was determined in simulation, but fine-tuned in Towne B2. We found a safe set of poses, in which the robot would stretch and reach over to the edge of the turntable, such that its links hovered above the y-axis of the base frame of the robot. The first pose sets the arm end effector directly above the train of blocks, almost creating a tunnel, while the second pose allowed for the end effector to pulse its gripper arms along the midpoint of the blocks. Since during testing we were unsure whether we would start the competition off by picking up dynamic blocks first or static blocks first, we made sure that these poses were safe for the robot to reach from its neutral position, as well as from the platform.

We made the assumption that blocks would have been moved and rotated by our opponent by the time we would pick up dynamic blocks. Thus, with this level of uncertainty in potential block positioning and orientation, we decided to let our block pulse at a random frequency. The pulse frequency was determined by `random.uniform`, which selected random durations for the gripper to be open. The wait times were floats between 0.5 s and 2.5 s. This allowed us to increase our chances of picking up blocks that had been moved by our opponent. We had determined that with a fixed frequency, it was possible to always miss a block. The challenge with obtaining the correct frequency was that, depending on the orientation of a block, if a gripper arm was closed too early, the block would rotate by around 45 deg, which made it very difficult to re-grasp the same block. The chance of having this block drop off the table was very high once a block was rotated by the gripper arms.

For making predictions, we had dug around the code stack to find out how `time_in_seconds()` was being computed, so that we could find `rospy.rostime` returned in nanoseconds. With this time, we made the assumptions that the time returned by `rospy.rostime` was accurate enough for angular velocity (and thus block position) predictions and that the motion for a block in a very small time (on the order of 5 ns) was linear.

We used `get_gripper_state()` to determine when the gripper arms sensed a block was grasped. Since gripper forces were not returned in simulation, we used the gripper arm positions instead, since this seemed to work in sim and real. If the gripper arm positions were greater than the gripper arms commanded 0.5 s after closing, the robot is commanded to stop pulsing and stay grasping, and move back to its hover position and then to safe intermediates poses towards the platform.

3 Evaluation

For the evaluation of our system we considered the success rate of four scenarios: first, the stacking problem with no reorientation, second the stacking problem with reorientation, and third, on picking up dynamic blocks, and finally the system integration as a whole. The success rate was determined by multiple factors which are highlighted by the list of metrics in each of the following sub-sections. The success rate was

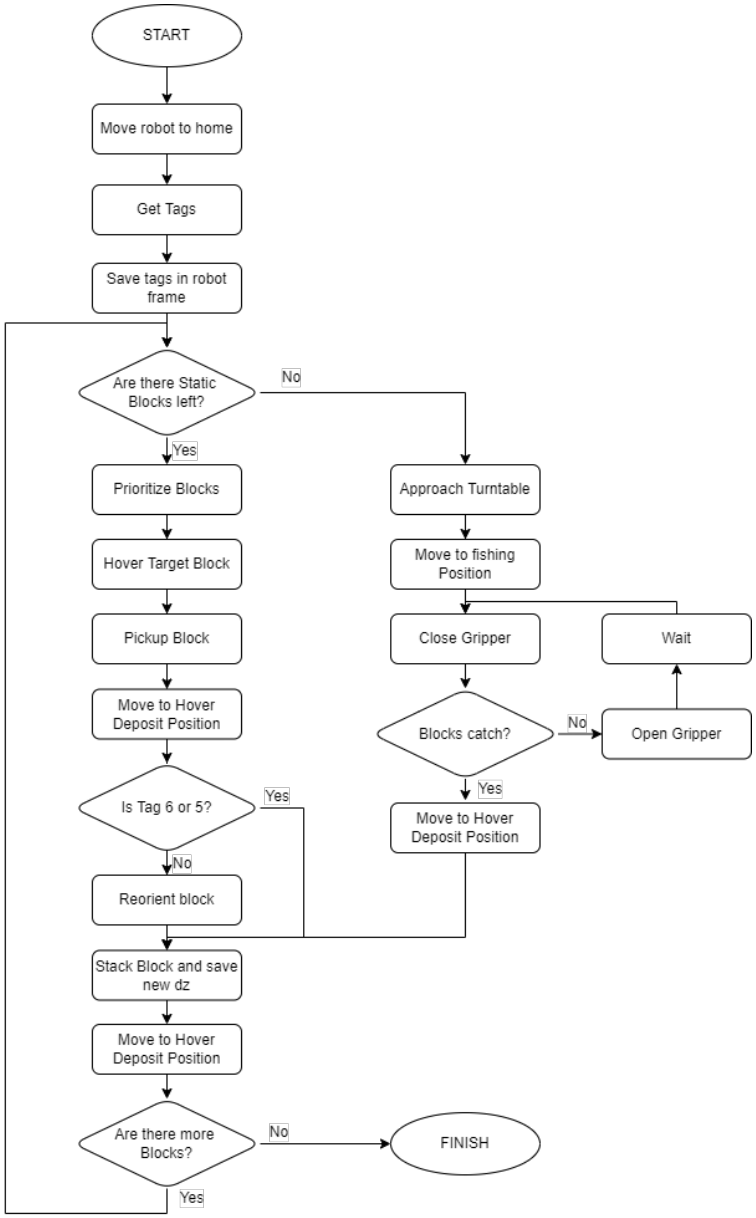

Figure 3: A complete flowchart of our approach. Each block represents a part of the program we have written, in each case. It is important to mention that any movement of the robot was validated in simulation.

determined by summing the number of conditions met and dividing it by the total number of items in the metrics list. Success rates were averaged across 5 runs of simulation. We worked in an incremental manner, starting with each scenario, and developing the next one only after the prior scenario was successful. We were only comfortable with proceeding to the next scenario when the success rate was at least 80% across 5 runs of simulation. We were confident that with more simulations and testing in the Towne B2 laboratory, in parallel to working on the new tasks, we could get our code to work at a 100 % success rate before the competition. We took this approach because we prioritized safety of the system throughout the development of our code, which required thorough experimental evaluation and debugging. It also meant that we had a solid foundation for our code.

3.1 Static Blocks Evaluation

We use an approach to develop small functions to be able to test each one independently. After checking each function, we were able to test them together. In this case, the evaluation for this scenario was simple. Following the metrics below, we were able to make sure of correct transformations into the robot frame. We evaluated and continued debugging and tweaking our code wherever necessary until we obtained repeatable, safe, successful picking capabilities in simulation. Once that was done, we took only one block at a time and placed it on the target position on the platform. Then we programmed a loop to increment the height of the tower to keep staking the static blocks. In simulation without reorientation, once we obtained success rate of 100 %, we then proceeded to test in the laboratory, where we were able to test this algorithm two times - it worked flawlessly!

1. Correct homogeneous transformation
2. Safe q condition 1: Not reaching joint limits (especially joint 6)
3. Safe q condition 2: Not obtaining "error -4"
4. Safe q condition 3: Not hitting table
5. "Success" from `solveIK` for q
6. Stack n blocks

3.2 Prioritization Blocks Evaluation

For the priority of picking up the blocks, we used a priority function that returns the tag name and the tag pose according to the order of importance given by the `order` defined above. As shown by the list of metrics below, first evaluated the success of this function by running print statements, returning the tag name and pose of the proposed order. Then, when the order seemed correct, we ran it on the simulation, to visually check whether blocks would be picked up in the desired order.

1. Correct order returned by function
2. Visually inspecting order of blocks picked up in simulation

3.3 Re-orienting Static Blocks Evaluation

The orientation of the blocks worked exactly as devised in the methods part. The only difficult part was to choose an angle for the robot end effector orientation that did not collide with the other blocks or get any joint angle errors.

We had a lot of difficulties when mirroring the code used in the red side to the blue side. Using the same position as used in the red except making the 'y' position the negative value as seen in x, resulted in many collisions and joint angle violations and sometimes even hitting the table. So, we have dedicated a lot of time to find the position on the table where the placement of these boxes did not gave these errors and collisions. One (0.5, -0.24) was found where there where the errors and collisions where obsolete. The blocks stacked successfully, when using this position. The position (tower position) of placing the blocks was (0.5, -0.24).

For the decided tower position, unlike on the Red side, on blue side the joint 5 of the arm would hit the target platform while placing the 1st block. To avoid this, we increased the safety distance parameter such that the arm does not touch the platform while placing the 1st block.

Below are some more observations made from evaluating the methods for reorientation of the blocks.

1. Correct rotations
2. Safe q condition 1: Not reaching joint limits (especially joint 6)
3. Safe q condition 2: Not obtaining "error -4"
4. Safe q condition 3: Not hitting table
5. "Success" from `solveIK` for q
6. Stack n blocks
7. All white face up

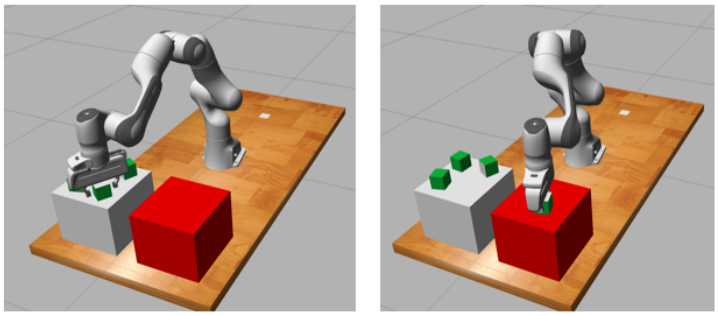

Figure 4: Panda arm demonstrating successful poses picking and placing blocks with top face `'tag6'` or `'tag5'`.

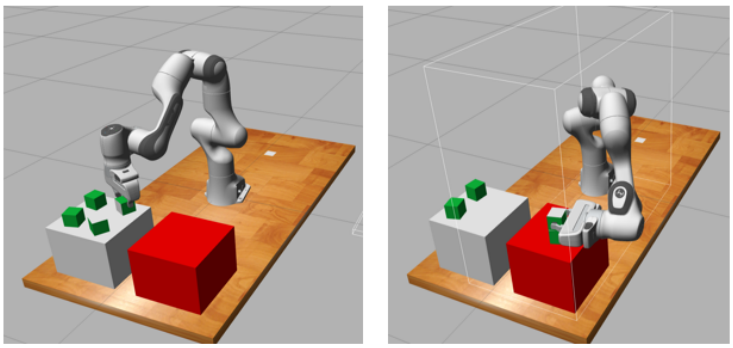

Figure 5: Panda arm demonstrating successful poses picking and placing blocks with top face `'tag1'`, `'tag2'`, `'tag3'`, or `'tag4'`. Note, blocks have been re-oriented with white face up

3.4 Dynamic Evaluation

To experimentally evaluate functions used for dynamic blocks, we had spent hours in simulation. Unfortunately, we did not have a lot of time to spend evaluating our code much in Towne B2, since this was one of

the last scenarios we had worked to test. That said, during the little in-person testing we had for dynamic blocks, we managed to pick up a dynamic block on our first try, during testing in Towne B2.

Similar to our methods for evaluating the success for picking up and stacking static blocks, we first started off with multiple tests to determine whether poses we were feeding were safe for the robot to travel to. These poses included (1) from the platform table to the turntable, assuming we were to start with picking up static blocks (2) from turntable to platform, assuming a block is picked up and needs placing (3) from neutral position to the turntable, assuming we were to start with picking up dynamic blocks, and (4) from the platform to the set of static blocks in their initial positions.

Then we evaluated our block prediction function. For making predictions, we had made the assumptions that the time returned by `rospy.rostime` was accurate enough for angular velocity (and thus block position) predictions and that the motion for a block in a very small time (on the order of 5 nanoseconds) was linear. To determine the validity of these assumptions, we ran simulations to test whether the block was able to move to the table, position itself above a desired block, and based on the time elapsed from hitting "ENTER", successfully close its gripper once the block was directly underneath the gripper arm. Since we knew we would implement a pulsing motion, we changed our evaluation approach to fit a setting that was closest to the competition. We knew that our evaluation methods in simulation would not have been as good as in-person evaluations, but by this point, we did not have enough lab time to test in the lab. Instead we changed the simulation environment to have blocks on the turntable edge.

The next step in evaluation was to check whether the robot had successfully sensed a block. This would determine whether the condition we used to terminate the pulsing motion was satisfied once a block was between the gripper arms. Since gripper forces were not returned in simulation, we used the gripper arm positions instead. When the gripper arm positions were greater than the commanded positions 0.5 seconds after calling the `arm.exec_gripper_cmd()` function, a block was deemed "sensed", and the robot arm would move to the hover position, and finally to the platform. A pick up was deemed successful when the robot arm completed these motions with a block in its gripper arms.

1. Correct termination condition for identifying a block between gripper arms
2. Safe q condition 1: Not reaching joint limits (especially joint 6)
3. Safe q condition 2: Not obtaining "error -4"
4. Safe q condition 3: Not hitting table
5. "Success" from `solveIK` for q
6. Timing
7. Number of blocks picked up in 10 minutes worth of simulation time (corresponding to roughly 3 minutes in real time)

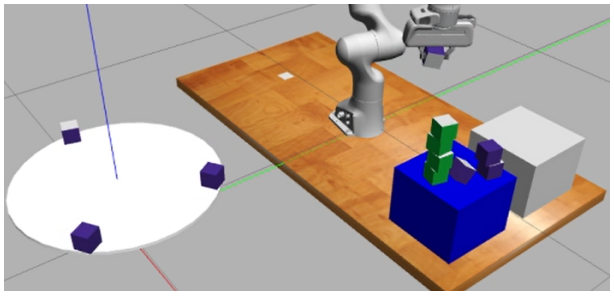

Figure 6: Panda arm demonstrating successful picking up and placing of dynamic blocks. Note that the placement is uncertain because in simulation, even if points of contact between end effector gripper arm is not perfect, blocks can still be picked up. This could lead to an untidy stack.

3.5 Systems Integration

After validating the success of each sub-module of our code, we then moved on to testing our system as a whole. One challenge was ensuring that between tasks, we would not cause blocks placed on the platform to fall. This involved checking all intermediate poses, ensuring that they would not interfere with blocks. We were very conservative with the way we had planned to stack blocks. Since we could not guarantee that the gripper arm could accurately sense the presence of a block, we had planned to place our dynamic blocks on a separate stack to the static blocks. This ensured that in the event that a dynamic block was falsely sensed, a block was not dropped onto the stack from a great and dangerous height, potentially causing the stack to topple over.

Thus, the first part of the systems integration involved determining safe locations for dynamic blocks to sit on the platform. Safe meant that (1) the robot would not hit the stack of static blocks when dropping off dynamic blocks, (2) the robot would not hit the stack of blocks when proceeding back to the turntable for more blocks, and (3) even if dynamic blocks had not been picked up perfectly, i.e. had been grasped at the block edge, the dynamic block would not touch the static block.

The second phase of integration involved making sure that it would be easy to switch the order of our tasks: picking up dynamic blocks vs static blocks. In case that during the competition, we wanted to switch our plan of attack, we ensured that making these switches would be easy. This phase included ensuring that while we kept track of the number of blocks being picked up, the robot could achieve required heights for stacking blocks. This was necessary in the event that we had decided to use our competition time to opt for only dynamic blocks, and the robot had the luck and time to pick up a large number of dynamic blocks.

Finally, we spent a lot of time going back and forth between the red and blue team to ensure that our code worked perfectly, and as expected for either scenario.

1. Stacks all static blocks
2. All white face up
3. Safe q condition 1: Not reaching joint limits (especially joint 6)
4. Safe q condition 2: Not obtaining "error -4"
5. Safe q condition 3: Not hitting table
6. "Success" from `solveIK` for q
7. Stack n dynamic blocks
8. Not knock down stack

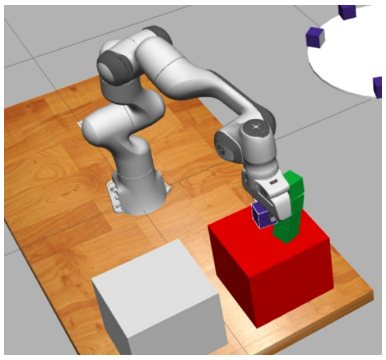

Figure 7: Panda arm hitting the stack of blocks due to poor choice of positioning of dynamic blocks on the platform.

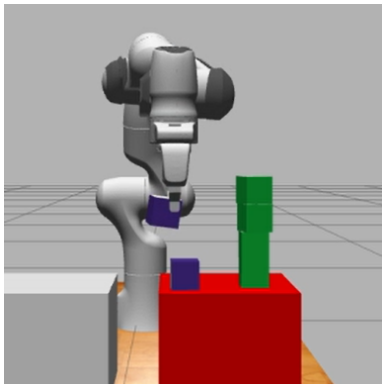

Figure 8: Panda arm positioning dynamic block at safe location on platform, far away from static block stack, regardless of gripping block at block edge.

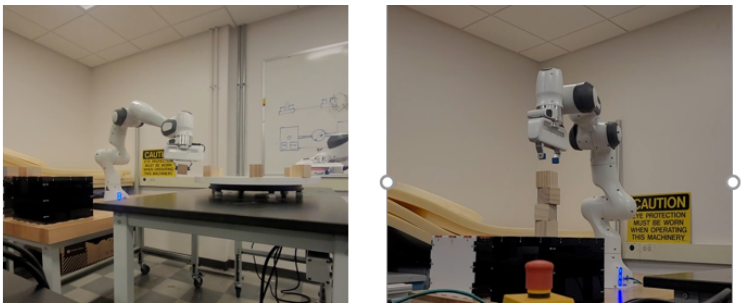

Figure 9: Panda arm successfully picking up dynamic block and stacking static blocks respectively!

4 Analysis

4.1 Static Blocks Analysis

This was the backbone of the program, we needed to make sure this part worked fine before continuing with the next steps. The main problems we face here were: First the generation of intermediate poses that can be used as seeds for our algorithm because the Inverse kinematic solver does not guarantee convergence. This was overtaken by minimizing the change of poses and using each generated pose as a new seed for the next step, also in the cases that we need to move the robot far from a desired point we use intermediate given (known) poses close to the desired final pose that we wanted. Second, the correct alignment of the end effector with the static blocks was overtaken with the knowledge of rotations and Homogeneous transformations, calculating a mirrored orientation for the robot end-effector with respect to the target block, this approach worked very well in simulation as well as in the laboratory.

4.2 Prioritizing Block Analysis

For the competition we have essentially considered tag5 as tag6 so that two rotations of tag5 is removed making only the tags from 1 to 4 and 6 getting the white face up while the tag5 gets stacked in an orientation that has been picked picked up by the end effector. This makes the stacking up of the blocks faster, which is helpful in getting around to picking the dynamic blocks at an earlier time.

4.3 Re-orientating Block Analysis

The robot was picking the blocks according to the priority given, every single time in simulation. However, when we actually tried to perform this in the lab there was an erratic behaviour of the robot. We found that unlike in the simulation, where we get the pose of the 'tag0' first followed by static tags and the dynamic

tags, the pose of the 'tag0' in the lab was not the first tag. So, we had to modify our code where the 'tag0' was taken first from all the given tags and then the remaining tags were taken and transformed into 'tag0' frame.

Furthermore, we implemented and evaluated two different approaches for picking up the blocks and reorienting them. One with picking and orienting the blocks vertically and the other with picking and orienting the blocks at a 45 degree angle. Picking up the blocks at a 45 degree angle made the reorientation more flexible, essentially making the area for placing the blocks a lot wider.

We have rerun these cases, of picking blocks vertically and at 45 degrees, individually, multiple times and found the one with the 45 deg picking approach had significantly more joint violations and collisions with the blocks than the one where the blocks were picked vertically. The one with the vertical picking approach had essentially 0 joint violations and collisions in all of its cases. Hence we have chosen an approach where the blocks are picked vertically and then reoriented when placing it onto the placement zone or the box.

When, we finally employed the case where the blocks where picked vertically and reoriented as seen in the evaluation, we found the joint violation error to be negligible. We only found the joint limit error(and only on joint 6) once every 30 times of rerunning the code. In order to make sure this does not happen, we iteratively added pi/2 when the joint angles exceed the upper bound of the joint limits and -pi/2 when the joint limits are below the lower bound. So this happens occasionally, but rarely, makes the orientation that does not get us the white face up.

On the Red side, initially, a safety distance parameter of 10cm was used in the `place_block()` function which caused the tower to collapse after stacking 3 blocks and moving to the pickup zone due to the difference in the heights of intermediate poses at pickup and drop locations. This was tackled by increasing the safety distance parameter to 15cm avoiding the joint 5 to collide with the tower when moving back to the pickup zone intermediate pose.

Furthermore, for the blue side, we increased the safety distance parameter such that the arm does not touch the platform while placing the 1st block. This caused the consequent blocks also to be released from a greater height than that on the red side. Although this was a bit scary during the simulation and the actual competition, it perfectly worked as expected with a minor issue being the triangular paper attached to the blocks during the competition, the diagonal edge of the paper would sometimes get stuck to the gripper and with the increased drop height on the blue side, there was a fear of improper placing of the blocks leading to a very unstable tower or a collapse of the tower itself.

4.4 Dynamic Blocks Analysis

During the evaluation of our functions for picking up dynamic blocks, we encountered some differences in simulation that we had not noticed on the real robot. For instance, the gripper arm took a long time to open in simulation. When wrapping the `arm.open_gripper()` code with print statements for time, we noticed that it took roughly 5 seconds for the gripper arm to open. Since we were using a pulsing method for the gripper arm (to sense a block and to increase the chance of sensing blocks that had moved from their initial positions), it became very difficult to time the robot to close its gripper at a desired time. When using the `arm.exec_gripper_cmd()` function alone, the gripper was much faster at closing.

When the gripper arm was held open for a random time interval, the robot was able to pick up more blocks in a given time. This behavior was confirmed in simulation. In reality, given our time constraints, we had only tested a fixed gripper arm open time interval. This observation likely occurs since with a constant wait time, the robot could always miss a block. However, it was evident that in using a random time interval, the problem of the gripper arm closing too early still existed, thereby causing the block to rotate. Once a block was rotated to a certain degree, it was almost impossible to pick up that block, because the gripper arms do not open wide enough to grasp onto the slippery wooden surface. In hindsight, it was clear that we should have either focused on the following (1) assume blocks will not have moved from their initial position, (2) make a good estimate of the table angular velocity during lab time in Towne, and assume it is the same angular velocity during competition, (3) predict the position of blocks on the table, and (4) make the gripper arm close at a point in time when it is certain a block is underneath it, based on the above assumptions.

In simulation, on average 5 times out of 8, the robot was able to pick up blocks on the table, without dropping them or knocking them off the table. However, we noticed that the way in which blocks were grasped deviated. This caused us to take a more conservative approach during system integration, i.e.

assuming that blocks could be on the very edge of the gripper arm. To add, in simulation, the robot was able to pick up blocks by holding onto what looked like the very corner of a block. It seemed as though the friction coefficient in simulation was much higher, since in reality it was much harder for the arms to get hold of blocks. In reality, it was more common for blocks to get rotated by the arms by about 45 degrees before getting picked up.

During evaluation, we noticed that the table angular velocity was very sensitive to how the robot arm interacted with the blocks. It appeared that the weight of the arm slowed down the table. Sometimes the arm had even stopped the table, or upon lifting a block away, the turntable gained speed. To make simulations closer to reality, we edited the way positions in which blocks spawned in simulation.

4.5 Systems Integration Analysis

While integrating all of the different tasks: picking, placing both dynamic and static blocks, it was evident that we had to weigh the importance between gaining points, working with the time limit, and providing safe motions. As usual, we prioritized safety, which meant conservative intermediate poses which were very safe in terms of (1) robot joint limits and (2) making sure no stacks were knocked over. This took extra time for the robot, however our evaluations deemed this necessary.

We also observed that our solutions were not exactly mirrored for both the red and blue team, because the robot design is not mirrored. Due to our methods for re-orienting blocks, it was clear that for the blue team, we had to make sure there was plenty of space between the robot arm and the static blocks sitting in their original position, to ensure that the arm would not crash into them. For the red team, this was less of an issue, because there were no blocks to the right of the platform, which is where the robot arm would reach across.

5 Conclusions and Lessons Learned

One of the main lessons learned during the preparation of this competition is that there is a significant difference in what is expected in simulation versus reality. In reality, we had expected that whatever pose information we were getting from blocks in Towne B2 would be in the same format as that in simulation. However, specifically for `'tag0'`, only upon testing our code in Towne B2, did we find out that this was not the first element in our tags list. This point is also linked with the way we had programmed, as this problem would not have come up if we had dealt with our code in a more general way, dealing with testing against all edge cases thoroughly, or not making any assumptions in the format of data provided. Other sim2real gaps that we experienced: (1) the robot motions were much smoother in real life than in simulation, (2) the gripper arm opened a lot slower in simulation than in reality, and (3) the robot seemed to take a long time to compute poses with `solveIK` and `calculateFK`.

In terms of the competition, in hindsight, we could have picked up dynamic blocks first for more points. In this way, the probability of picking up a block would have been higher, since we would have made less assumptions about the re-positioning of blocks after being touched by other teams. Dynamic blocks clearly provided a significant amount of points. We could have sacrificed some safety concerns in simulation earlier on in our work, so that we could focus on tasks in parallel with one another.

If we were to repeat this process again, we would have made sure to spend adequate time on the real robots as early as possible; also, we would have developed solutions conceptually together before programming just like we did, since that seemed very helpful for us.