# MEAM5200 Final Report

Sai Chand Gubbala, Rahul Birewar, Jessica Ling, Jalaj Shukla

December 21, 2023

## 1 Introduction

Pick and place tasks are some of the most relevant and important uses of robotic arms, with industry applications including assembly line automation, food packaging, and object sorting. In this report, we describe the methods with which we implemented an automated pick-and-place algorithm for static and dynamic objects, as well as our evaluation and analysis of the performance of this algorithm on the 7 degrees of freedom (DOF) Franka Emika Panda arm.

## 2 Methods

In this section, we will describe the overarching game strategy and the code implementation of our algorithm. We will first discuss our strategy, which forms the basis for our approach to state estimation and block-picking actuation. Then, we cover our approach to and assumptions made for state estimation, or the state of the environment at any time, before describing the procedure for block picking for static and dynamic blocks, and for block stacking.

### 2.1 Strategy

The pick and place algorithm we implemented was designed for the scope of a block stacking competition, in which the score of each block was based on its type (static or dynamic) and the height of its center from the goal platform. We quickly determined that the optimal method for gaining points would be to stack the blocks; if all four static blocks were directly on the goal platform without any stacking, with an individual value of 10 and height of 25mm, this would lead to a cumulative score of 1000. On the other hand, if even one dynamic block was stacked on top of one single static block, this would lead to a cumulative score of 1750, and if all four static blocks were stacked in one tower, this would lead to a score of 4000.

The score itself was not the only factor to contend with. With a time limit, we would need to consider the amount of time to complete a single task, that being the picking and placement of a single static or dynamic block. We decided that we would prioritize consistent performance on the more predictable task, the static blocks, to guarantee a score of 4000 points. Then, because of the height of the next block, stacking at least one dynamic block would more than double that score, to a total of 8500 points. With uneven lighting conditions at the arena we decided on initially stacking the static blocks and then picking up the dynamic block rather than stacking it directly on the tower we dropped it on the platform, reconfigured its position, and then picked it up analogous to the previously picked static blocks. It increased the dynamic block stacking time but it even guaranteed precise placement.

Our main secondary task was to avoid collisions, which would automatically disqualify us from the competition. As well, we prioritized side-on collision avoidance with the block stack, which would not disqualify us but would greatly decrease the block height of any subsequent blocks, lose established points, and remove blocks from play entirely as they would fall below the 0.2m height limit.

### 2.2 State Estimation

State estimation is a critical aspect of autonomous robot arm control, particularly in a setting involving dynamic objections and requiring collision avoidance and feedback control. Estimation of the environment, including target objects and monitoring of the robot's own condition allows autonomous robot control to dynamically respond to changes in the environment.

In the scope of this project, we were able to make several simplifications and assumptions to reduce the complexity of the state estimation task. Firstly was that the simulation environment was a mostly accurate and consistent representation of the setting of the competition; the location of the robot and block platforms could be assumed to be consistent relative to the world frame origin. Secondly, we assumed from the safety specifications that nothing would be introduced into the robot workspace that wasn't in the problem.

Therefore, the only variable in the environment besides the arm itself would be the locations of specific blocks, and of these, only the dynamic blocks would change over time. To estimate block locations, we used the given `get_detections()` method from `ObjectDetector()` to obtain the $4 \times 4$ transformation matrix for any block detected within the current field of view of the camera, $T^{cam}_{block}$. The transformation matrix would give the position and orientation of the block relative to the camera; we used another given method, `get_H_ee_camera()` to obtain the transform from the camera to the end effector, $H^{cam}_{e}$.

To utilize these block detections, we find the end effector pose with respect to the robot base frame by using the `getState()` method of the `ArmController()`, which yields the transformation matrix $T^0_e$.

$$T^0_{block} = T^0_e \,(H^e_{cam})\, T^{cam}_{block} \tag{1}$$

In the simulation, the detected transformation matrices matched the block positions exactly. However, the April Tag detection in hardware introduced noise into the transform matrix. We employed a strategy where we aligned the end effector first along either side of the block so that the gripper could hold the block properly on both faces. For this, once the pose of the block is determined w.r.t robot base frame, we find out the axes/direction cosines that are *not* parallel to the world z-axis. The blocks' axes are randomly oriented and the z-axis of the block is not always pointing in the z-direction of the world. Once we determine the axes/direction cosines in the pose matrix of the block that is in the world x-y plane, we simply use one of those axes for aligning the end effector with the block. Further, we imposed a constraint that the z-axis of the end effector is always pointing in the negative z-axis of the world. Hence by simply doing a cross product of the previous two axes, we could get the third axis in the orthogonal triad which then completes the rotation matrix for the end effector in the robot's pose matrix.

Once we calculate the pose of each detected block in the world frame, we can then solve the Inverse Kinematics (IK) problem, which we will discuss in the following sections.

## 2.3 Picking Static Blocks

For picking the static blocks, we want to adopt an optimal strategy that minimizes the time for overall picking and stacking the static blocks. So we employed a strategy where the robot goes to a scan position where the camera detects all the static blocks on the base and gets the position and orientation of the static blocks using the transformation from camera to end effector and the transformation from the end effector to the base frame of the robot. Since we did this scanning step only once, we had to experiment with a lot of different poses that gave robust detection both in the simulation and the real-world testing. We did this process for both the red and the blue robots.

Once the detections were converted to transformation w.r.t robot base frame, the robot goes to a pre-picking pose where the end effector aligns itself according to the orientation of the block. Initially, we skipped this step to improve the speed of picking but we observed that the robot spans the rotation from the current position to the end position. So the end effector was colliding with the block that it was going to pick. Hence we introduced the previously mentioned pre-picking pose to ensure the end effector aligns first with the block. Then all the end effector has to do is lower down to pick up the block and close the grippers to pick it up. Here we use the pre-pick configuration as the seed configuration for the picking pose to ensure faster gradient descent convergence. Finally, we added another intermediate step where the robot's end effector goes to a central position but at an elevated Z height of about one block length. This step is necessary to avoid the end effector knocking off the other blocks on the base when it is going from the base to the red or blue base.

## 2.4 Picking Dynamic Blocks

The task of picking dynamic blocks was significantly more complicated than picking static blocks. In this section, we briefly describe two initial methods we attempted before a detailed description of the method we eventually decided upon.

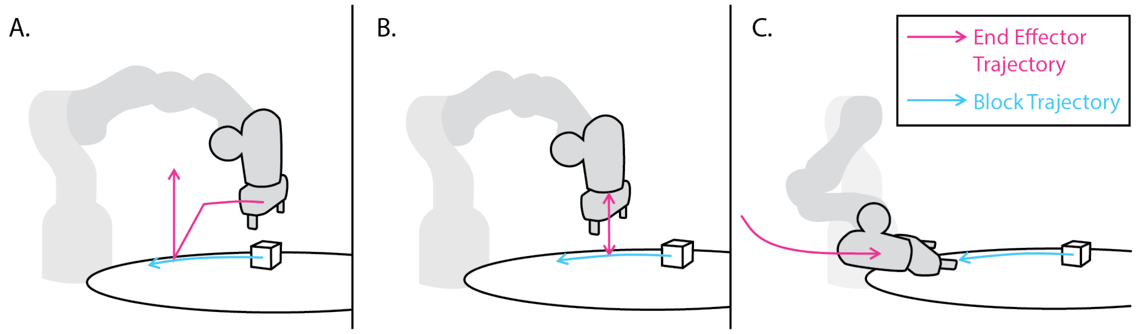

**Figure 1:** An overview of the different approaches we attempted for picking dynamic blocks, C being the method we ultimately decided on. (A) The ”follow-and-intercept” approach, where the end-effector follows the trajectory of the block and moves to intercept it once its orientation matches the predicted orientation of the block. (B) The ”wait-and-intercept” approach, where the end-effector waits at a set position and follows a linear interception trajectory. (C) The ”blind intercept” or ”fishing” approach, where the end-effector waits in the shown position with open grippers and assumes a block will end up in it, which it detects by checking gripper positions after attempting to close them.

### 2.4.1 Approach & Assumptions

We realized early on that we *could* follow a similar strategy to the static blocks, in which we first move to a position to view all of the dynamic blocks, from which the calculation of the block radii (defined as the horizontal distance of the block center to the world z-axis) would be straightforward, as described in section **2.2**. However, doing the block surveying step is inefficient for two reasons. First, if we only do it once at the start, as we do with the static blocks, then we have no way of knowing if any of the blocks have changed position or moved entirely, whether due to the opponent's actions or to our own. Second, if we do it at every attempt at getting a dynamic block, then we waste time, a very finite resource in the competition setting.

State estimation is particularly important for dynamic blocks since these blocks change position both as a default and as a result of the game antagonist's actions, such as sweeping blocks off of the table. To simplify this, there were some assumptions that we could make as well.

First, we assumed that the radius of any block we wanted to target would be within a certain range (approximately between 0.22m and 0.28m) from the world z-axis. Outside of this, the block would have fallen off of the dynamic platform ($> 0.28$m) or led the robot to be close to a singularity ($< 0.22$m). We also identified the angular velocity of the rotating platform to be roughly $\omega_{plat} = \frac{2\pi}{100} \approx 0.063$ rad/s and assumed this to be constant.

### 2.4.2 Attempted Strategies

Our first attempted strategy was a theoretically robust approach implementing inverse velocity kinematics and computer vision feedback for block trajectory and orientation. The overall strategy can be described as the end-effector following the circular trajectory of the block, which would be identified by positioning the end-effector at a known, set position, and then adding a vertical component to the end-effector trajectory to intercept the block. The approach is illustrated in **Fig. 1A**.

This strategy was based on the intuition that, if one were to accurately have the end-effector follow the block trajectory, then the block *position* (if not orientation) would be stationary with respect to the end-effector. However, this would either leave unaddressed or complicate many other aspects, including (1) how to match the end-effector orientation with the block; (2) time-syncing the end-effector with the block in the first place; (3) handling event logic, e.g. when to give up on the current block or which block in the current field of view to choose.

The second attempted strategy, illustrated in **Fig. 1B**, attempted to simplify the approach made in the "follow-and-intercept" method, choosing instead to do a simple vertical motion, timing it such that it closes the grippers exactly as the block passes through them. However, we found that this approach was inconsistent even in simulation, due to inconsistent timing and imprecise end-effector positioning, and would likely not be robust enough to work on hardware.

Ultimately, the biggest problem with both of these approaches is the fact that they approach from the top down. We were not able to guarantee that the grippers wouldn't collide with the top of a block, especially if there were two or more blocks clustered closely together, which would result in a disqualifying collision. The risk was substantial enough that we decided to move on to a different strategy altogether.

#### 2.4.3 Final Strategy

Our final strategy was a simple "fishing" method where we always align the robot in a pre-determined pose that's ready to pick up the block. For this, we iteratively determined a pose that can place the end effector of the robot in an orientation and position that can easily pick up the dynamic blocks on the rotating table without colliding with the table itself.

The next challenge was to figure out a way to determine if a dynamic block was in between the grippers or not. As stated in the previous section, synchronizing time was a significant challenge and hence we opted against using any sort of vision-based confirmation to close the grippers. Instead, we simply close the grippers once the robot is in the pre-determined picking pose and get feedback. For the feedback mechanism, we simply implemented the `exec_gripper_cmd` command and gave 3cm as the distance between the gripper hands. This gives us two scenarios when we get the positional feedback from the grippers - (i) the distance between the grippers is 3cm, which indicates the block is not picked between the grippers; (ii) the distance between the grippers is 5cm, which indicates the block is picked between the grippers.

For scenario (i), we loop the robot to move back and then come to pick up position again and close the grippers to check if the distance is 5cm. Here we included an additional step of moving back and coming to the picking position as it proved effective against piling up blocks between grippers and also sweeping blocks that aren't on the periphery to the edge of the table (which is a requirement for our strategy to work!) For scenario (ii), we exit the loop and proceed to place the dynamic block on the base where we have static blocks placed. Then we go to a scan pose which helps us to accurately pick up the dynamic block and place it on the stack of static blocks on the blue or red base. The additional effort of going to the scan pose and picking the dynamic block proved to be very effective as anticipated as there's uncertainty involved in the orientation in which we pick up the dynamic block in the fishing pose.

## 2.5 Stacking Blocks

The strategy for stacking the blocks was to place the blocks one over the other in a single tower. Our approach was based on finishing up simple tasks first and maximizing the points within the limited amount of time. Hence we opted to go for the stacking of all the static blocks first and then proceed for stacking the dynamic blocks on the top of the stack of static blocks previously mentioned.

We hard-coded the seed configurations required for solving the joint configuration that corresponds to a particular level in the stack (level 0, 1, 2...etc). This helps the gradient descent method to converge faster. For every block we place on the stack, we simply add a block height to the z-value in the target pose for the IK solver and get the corresponding joint configuration to place the block. Once the robot executes this step, we use `open_gripper()` to release the block from the grippers. We considered a tolerance/offset when updating the z-value in the target pose so as to not run into a collision warning with the existing stack. With the offset provided, the end effector only goes until a certain distance (1 cm) between the new block and the existing tower and then opens the grippers.

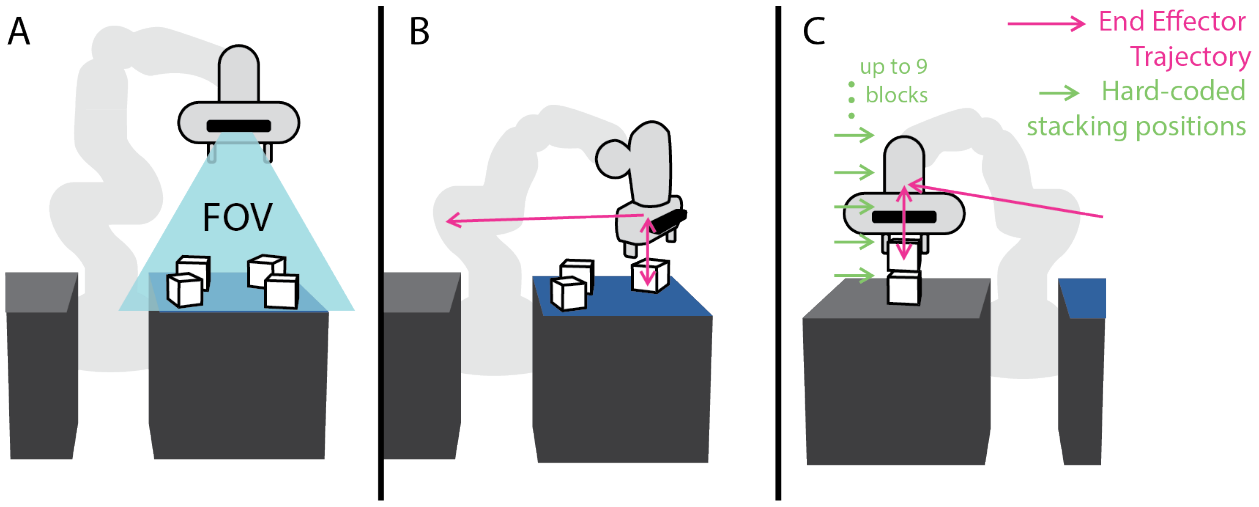

**Figure 2:** An illustration of (A) the viewing position used to scan for all static blocks. (B) the end-effector matching the orientation of a target block before picking it. (C) the hard-coded placement positions and the procedure for placing a block

One caveat of this strategy is that if, for some reason, a particular block is not picked during the static block stacking, the end effector still proceeds to the placing pose on the stack even if there's no block in between the grippers. This results in the end effector releasing the next newly picked-up block from a distance equal to one block height instead of the previously stated 1 cm. We were aware of this limitation and hence we experimented with a variety of scan poses and finalized the one that gives robust detection of the static blocks both in the simulation and the real-world tests.

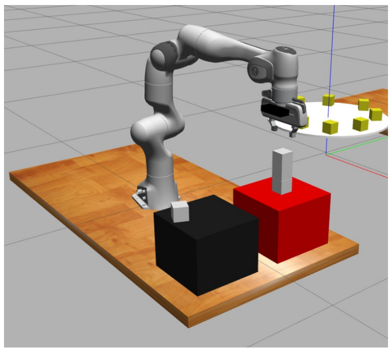

**Figure 3:** Caveat with stacking strategy where one of the four static blocks is missed in the detection phase and the robot still proceeded for the placing pose over the existing stack

## 3 Evaluation

### 3.1 Simulation

We used the simulation to evaluate the overall soundness of our approach. While there would be many differences between simulation and hardware execution that would need to be addressed, success in simulation was a prerequisite for success in hardware. Therefore, it was vital to assess the robustness of our approaches for both static and dynamic blocks on simulation.

To evaluate our static block pick-and-place, we evaluated the algorithm's objective ability to stack blocks, as well as qualitative measures. These qualitative observations included consistency of block placement, i.e. whether each successive block was centered in its placement to the target; risk of collision, i.e. whether the end-effector trajectory brought any part of it too close to obstacles, blocks, or the block tower; and whether there were any extraneous movements that could be eliminated.

Our approach to evaluation in the simulation was to leverage it to rapidly iterate upon our algorithm, using qualitative observations to quickly refine and debug. Some examples of these are provided in Table 1.

| | Observation | Problem | Solution |
|---|---|---|---|
| 1. | Gripper over-rotates to match the exact orientation of static block | Causes extraneous motion & can cause slow or no IK solution finding | Add 90-degree rotation redundancy |
| 2. | Gripper rotates while moving down to grasp static block | Gripper arms can collide with corner of block, resulting in collision error or moving the block | Decouple rotational and gripping motion; match block orientation before performing vertical movement |
| 3. | Static blocks are being pushed when the gripper is bringing a block to the target platform | Violates static assumption and can push blocks out of play | Add vertical displacement step before going to target platform |
| 4. | End-effector can collide with dynamic block and wedge it between itself and the platform | Results in collision error | Reduce vertical displacement between end-effector and platform |
| 5. | Opening and closing grippers while end-effector is stationary can push dynamic blocks out of the way | Results in longer waiting time and decreases likelihood of getting a block | Repeat sweeping motion to grasp block |
| 6. | No dependable way to grasp dynamic blocks in an orientation compatible with stacking | Directly stacking dynamic blocks after grasping from moving platform results in falling blocks and tower destabilization | Put dynamic block on static block platform and use static block detection/picking algorithm to stack dynamic block |

**Table 1:** Some examples of iterative improvements made on our static approach (1-3) and our dynamic "fishing" approach (4-6) by evaluating them using simulation.

## 3.2 Hardware

Evaluation of hardware was important to the success of our algorithm because thorough testing was vital to understanding the particularities of the environment that would prevent an algorithm that worked on simulation from working in the competition. Therefore, it was important to document and understand the cause of any unexpected behaviors in hardware testing.

Our overall approach in testing on hardware was to run the procedure we already verified on the simulation. We would evaluate performance on hardware by analyzing how much time it takes to pick and stack static and dynamic blocks, the success rate of static and dynamic blocks respectively, and, if failed or exhibiting unusual behavior, identifying what aspect of the algorithm did not perform as expected. The first two metrics are described in Table 2, and some instances of algorithm troubleshooting are described below.

A major component of hardware testing was determining any offsets that were different between hardware and simulation. We found that differences in just a fraction of an inch in the platform height could impact how low on the block it was grasped, which could result in collision with another block, as illustrated in **Figure 2A**. A different sort of offset in hardware was the different field of view of the camera, requiring viewing positions to be adjusted to ensure all four static blocks would be within the viewing frame of the camera while making sure no blocks on the goal platform would be identified and targeted.

Another important aspect of hardware testing was noise detection and handling. As mentioned in **2.2 State Estimation**, block detection in hardware involved noise including deviations in the z-axis, which we found from evaluation in the recitation lab space. We discovered on the day of the competition the degree to which light conditions would impact block detection. With a dimmer, less perfuse lighting, block detection became less dependable, whereas block detection in the recitation lab space was near perfect unless April tags were damaged.

Finally, timing and unexpected latency periods on hardware were vital to understanding performance in the competition. As shown in Table 2, the time per static block increased almost twofold when going from the simulation to the first hardware test. There was an unexpectedly long waiting period (sometimes up to 20 seconds!) between orienting the end-effector and moving it vertically to pick the static block, in which the algorithm was solving the IK problem for the roughly two-inch trajectory. We eventually found that changing the IK helper function by changing whose `lib` folder we were using solved this issue, resulting in a roughly 46% reduction in time spent per static block.

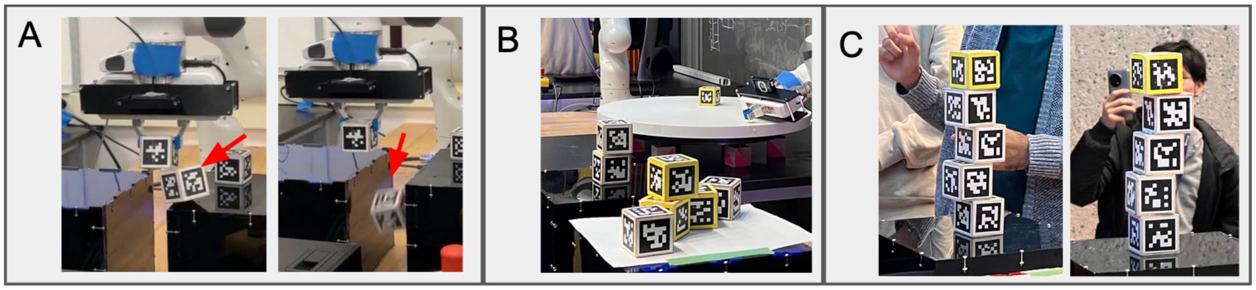

**Figure 4:** Examples of problems observed in hardware that were not present in the simulation. A: Z-offset of the platforms, resulting in the block being lower in the gripper than expected, pushing other blocks off of the platform. B: Incomplete detection of static blocks, leading to only two static blocks being stacked and failure of the dynamic procedure. C: Noise in detection of static blocks, leading to imprecise (off-center) static block picking, leading to unstable towers.

## 4 Analysis

| | | Static | | Dynamic (fishing) | |
|---|---|---|---|---|---|
| | | Time per block(s) | Completion Rate | Time per block(s) | Completion Rate |
| Simulation | | $19.8 \pm 0.837$ | 100% (n=5) | | |
| Hardware | Lib 1 Default speed | $35.6 \pm 11.06$ | 83% (n=6) | | |
| | Lib 2 Default speed | $22.25 \pm 0.5$ | 100% (n=4) | | |
| | Lib 2 Speed=0.3 | $15.94 \pm 0.73$ | 88% (n=18) | $43.6 \pm 0.89$ | 50% (n=10) |

**Table 2:** A summary of the performance of our algorithm on static and dynamic blocks on both simulation and hardware. Completion is defined as the successful picking and placement of a single block, and the completion time is defined as the time difference from when the robot initiates movement towards picking the block to when the robot releases the block onto the stack. Hardware trials are split by the different parameters under which the algorithm was tested, changing first the `lib` folder that was being used and then changing the arm speed parameter.

### 4.1 Detection of Static Blocks

We restricted ourselves to scanning the static platform once prior to picking static blocks. The positions and orientations of the detected blocks were stored with respect to the world frame and were then indexed when picking the static blocks. With multiple trials, we were able to find a scanning pose that was suitable for the camera to detect all 4 blocks on the base and accurately identify and store their specific values. Hence, we continued to scan just once.

Assuming no external disturbance during the interaction with other blocks led to an increased efficiency as this approach reduced time spent on repeated block detection. This also ensured a consistent reference frame for the robotic arm, in turn reducing the overall computational load and complexity of the system, making the operation smoother and more repeatable.

This assumption, however, is just an assumption. As observed in lab sessions, if the arm unexpectedly moved a block while picking up another, the change in a single measurement affects the performance due to the lack of adaptability. Additionally, this strategy heavily depends on the accuracy of the initial block detection and the quality of the seed values provided to the Inverse Kinematic solver. Any errors or inaccuracies in this initial step can lead to subsequent issues during block stacking and picking. This was the exact reason why we lost in the final matches at the competition. During the initial detection, the camera only detected 3 blocks. Without feedback present to rectify its error, the overall task decided for the arm collapsed. The lack of real-time correction results in inefficiencies and the arm misses opportunities to correct errors. It was evident during the competition when we stacked 4 static blocks completely aligned and with the addition of another block would have simply toppled the tower.

Possible solutions to these weak points include:

(a) Use Kalman filters to mitigate the issues with the detection of blocks due to varied and inconsistent lighting conditions, increasing the chances of detecting them more accurately.

(b) Introduction of intermediate checks at predefined intervals i.e. after every two blocks to update the overall status of the blocks allowing the system to correct any error before it accumulates.

(c) Taking multiple readings from various poses and then rectifying the values. i.e. rather than taking a single reading from a single predefined scan pose, we could take 3 different readings from 3 different predefined scan poses and then generate accurate values. It still assumes that the blocks remain undisturbed during the complete execution.

## 4.2 Aligning the End-Effector and Stacking the 4 Static Blocks

We leverage an Inverse Kinematic solver coupled with a gradient descent algorithm for the precise execution of pick-and-place tasks. The effectiveness of this method heavily relies on the accuracy of the initial seed values, as they serve as the starting points for the solver and significantly influence the robustness of the block-picking process.

To enhance the accuracy of the seed values, our strategy involves positioning the end effector in a pose just above the target block. In this intermediate pose, the grippers align themselves symmetrically to grasp the block near its center. This alignment ensures a more stable and centered grip on the block during the subsequent pick-up operation. Following the alignment, the end effector proceeds to grasp the block and then move toward the predefined center of the dropping platform for optimal placement. Before the actual placement occurs, the system transitions to another pre-drop pose. From this intermediate pose, the block is methodically placed rather than simply dropped. All in all, our approach involves performing four Inverse Kinematics solutions: initial alignment, block pick-up, movement to the dropping platform's center, and the final placement. Each of these steps contributes to the overall precision and reliability of the pick-and-place operation.

We combined these steps of alignment and movement together but it resulted in timing variation and hence had to stick to the modular intermediary steps method. This strategy follows a sequential approach, ensuring each step is executed in a well-defined order. With gradient descent, our main aim was to reduce as much variability as possible, hence employing this approach enhances reliability and predictability. It is also modular, breaking down the overall task into smaller, manageable components. This facilitates easy debugging, maintenance, and last-minute changes.

Using hard-coded poses, including a predefined dropping platform's center, provides clear definitions for specific states and positions that are collision-free either from the surroundings or from the tower. It also sped up the robot's performance by reducing computational needs. This clarity can simplify the development and debugging process, along the knowledge of the hard-coded dropping platform's center ensures a stable reference point for placing blocks, contributing to consistent placements. Unfortunately, dependence on hard-coded positions without feedback information about the current state of the environment limits the adaptability to dynamic environments or changes in the task.

Furthermore, since the overall code functions on the assumption that every task performed till now is error-free, this leads to placement errors if the actual count differs. As the blocks are dropped from a fixed height during placement, irrespective of the actual height of the previously placed blocks, potentially causing misalignment, which can be seen during several instances in the competition specifically in the finals. Overall, the strategy lacks real-time adaptation to changes in the environment, relying on predefined poses and assumptions only.

Possible solutions to these weak points include:

(a) Movement of certain joints can be restricted and predefined to reduce the overall calculation of poses for all the joints and improve the speed further.

(b) Implementing feedback mechanisms using the camera which would detect the position of previously placed blocks would significantly improve the system's understanding of its environment. This feedback could be utilized to make informed decisions during subsequent placing operations. It would even consider the height adjustments depending on the previously placed blocks.

(c) Rather than using the gradient descent for solving the Inverse Kinematics (IK), various alternatives such as geometric methods, could potentially yield faster and more efficient solutions.

### 4.3 Positioning by the Turntable and Grabbing the Dynamic Blocks Using the Fishing Method

From the 3 strategies discussed above, we adopted the fishing method where the arm would swoop in at the turntable in a predefined hard-coded pose with an aim to catch the incoming blocks by periodically opening and closing the gripper. The grippers would even perform a check confirming the presence of the block by checking the distance between the two grippers.

The predefined hard-coded pose and periodic opening and closing of the gripper provided a predictable and reproducible movement pattern, where it was guaranteed to operate safely adhering to all the criteria contributing to the stability of the strategy. As the grippers perform a real-time check by measuring the distance between them, we can ensure the presence of a block is always confirmed. This real-time verification enhances the reliability and even if it did not grab it earlier it would move back to the previously determined pose and then swoop in again allowing certain congestion to pass away which might get created during the earlier action of waiting and grabbing the block. While the fishing method did not always guarantee the immediate picking of a dynamic block, we were able to ensure we only continued if a block was picked with these checks.

Due to the hard-coded nature of the fishing poses, which were achieved after numerous trial and error methods, it is unaware of the environment, hence there is always a chance of collision if there is some change. In the competition's semifinal match, the robot arm collided with the top of a dynamic block as it came down into the fishing pose, causing it to stop. While moving to grab the blocks again and again there is a huge possibility of error accumulation and the arm drifting violently if the secondary task is not defined properly.

Possible solutions to these weak points include:

(a) Development of predictive models for block trajectories and rotations. By analyzing the rotational dynamics of the blocks, the arm can anticipate their future positions and optimize its catching strategy accordingly. Clubbing it by checking the position of the rotating blocks in real-time using the camera. The arm can then dynamically adjust its trajectory and gripper movements based on the observed positions and orientations of the block and then precisely pick it up all the time it detects. As it is explained in the strategy section for picking up the dynamic blocks

(b) To overcome the possibility of collision, it is mandatory to verify and test the hard-coded pose in simulations as well as on the hardware. Acknowledging the discrepancies between the simulation and the hardware it would be recommended to work with a tolerance factor and recheck the poses in the hardware with caution.

### 4.4 Reorienting the Picked Dynamic Block Prior to Stacking

After picking up the dynamic block we initially tried to place it on to the stacked tower but it always fails to do so. As grabbing the block with proper alignment is difficult making the overall strategy prone to erroneous placements of the blocks. Thus, we planned to first orient the block by placing it on the base platform and then scan its position and orientation. This addresses the alignment issue by first orienting the block on the base platform before attempting to place it on the stacked tower thereby always ensuring the precise grabbing and placement of the block without the risk of toppling the tower.

We then moved ahead to stack the dynamic block, treating it as a static block. Since this method was very robust for the static blocks, it made sense to use it again for stacking the dynamic blocks. As the height of the stack increases, the tower becomes much more sensitive to eccentric placements of the blocks. Therefore, reliability was critical.

In between the fishing pose and block drop-off pose at the static platform, we implemented hard-coded intermediate poses in order to move the robotic arm from opposing ends of the environment without colliding with the tower. For simplicity, these pose modifications of the start position. However, they were not the most efficient as this resulted in more total joint motion than was necessary. While this entire procedure did require a long amount of time to place a single dynamic block, it was very effective and always successfully placed the block on the tower.

A loophole with this approach was we placed a check in the code to see if the number of dynamic blocks detected was one or not and proceeded to pick only when the number of new detections on the base was one and if not continued to go for a new dynamic block (as we assumed zero detection in this particular scenario means either block was dropped while transporting to base or it fell off the base when the gripper release the block to drop on to base). We didn't consider all edge cases in this assumption and it resulted in an interesting observation during the competition.

Since we enforced to go for picking dynamic blocks only if there is one detection, it automatically rejects picking if there is more than one block on the base. During the competition, the camera detected only 3 blocks initially and proceeded to stack those static blocks. Then it proceeded to go for dynamic blocks and placed it on the base as mentioned in the previous section for dynamic picking strategy. But since the camera missed a static block detection and a new dynamic block was placed on the base, when the robot went for the scan position, it detected two blocks this time and, due to the check we enforced, the robot never went for picking and instead kept picking up the dynamic blocks from the rotating table and kept putting it on the base and the loop continued. This strategy could be improved further in future work.

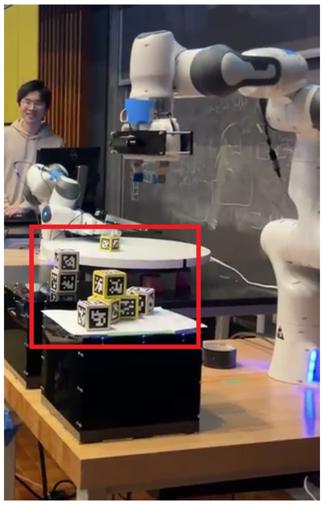

**Figure 5:** Erroneous picking due to loophole in the code

Possible solutions to these weak points include:

(a) Develop more efficient intermediate poses or incorporate a planning algorithm (potential fields or RRT) for obstacle/tower avoidance for improved speed.

(b) If only 3 blocks are detected when we know to expect 4, then try a different camera angle to attempt block surveying again. This would improve the chances of removing reflective noisy light from the background.

Apart from these issues, it was observed that the offsets for each of the bots were different and hence there was an introduction of steady-state error and offsets seen when the same code is plugged in the other bot. In the sections where hard coding was not implemented, it performed flawlessly such as block detection and its placement along with the alignment.

We also noticed an interesting correlation: the speed of the final robot's solution was influenced by the libraries used for calculating the Jacobian matrix, inverse kinematics, and other parameters. Consequently, we observed variations in the convergence speed of the same final code, depending on how the other libraries were defined, which was different for each one of us.

Finally, as mentioned briefly in **3.2**, the performance of block detection varied widely based on lighting, as we saw when comparing the success with static blocks between the recitation lab space, which had bright perfuse light, and the auditorium that the competition took in, which had a dimmer, diffuse light. Refinements could be made in the future to the April tag detection algorithm to improve that performance. In addition, it was a known bug that the grippers would sometimes fail to close or to open; our fix for this was to specify the force at which to close the gripper in `exec_gripper_cmd(pos, force)`, which resulted in consistent gripper opening and closing.

| | Static | | Dynamic | | | | |
|---|---|---|---|---|---|---|---|
| Round | # stacked | Time to stack | # stacked | Time to stack | Collapsed? | Score | Termination Reason |
| Qual 1 | 4 | 2:16 | 0 (1 queued) | — | No | 4000 | Time limit (Success) |
| Qual 2 | 4 | 1:41 | 1 (1 queued) | 0:57 | No | 8500 | Time limit (Success) |
| Quarterfinal | 4 | 1:34 | 3 | 3:22 | No | 22500 | Time limit (Success) |
| Semifinal | 4 | 1:32 | 1 | 0:45 | No | 8500 | Dynamic block collision |
| Final 1 | 3 | 1:11 | 0 | — | No | 2250 | Static block collision |
| Final 2 | 2 | 0:48 | 0 | — | No | 1000 | Dynamic block loop |

**Table 3:** A summary of the performance of our algorithm in the block-picking competition. In the Semifinal, the algorithm terminated as a result of a block collision detected while attempting to pick a dynamic block from the rotating table. In Final 1 and Final 2, not all static blocks were detected, leading to unpredictable and previously unseen behavior. Note that Qualification rounds had a time limit of 3 minutes, which was increased to 5 minutes in the Quarterfinals and beyond if either team scored above 4000 points.

## 5 Lessons Learnt and Future Steps

The block-picking task was a deceptively simple task that involved state estimation through computer vision and arm state feedback, collision avoidance, and optimizing the inverse-kinematics solver implemented in previous labs. Overall, we succeeded in the scope of the course, earning second place overall in the class-wide block-picking competition. We were able to achieve a maximum stack height of 7 blocks with a score of 22500, after refining our static block pick-and-place algorithm and going through three iterations of the dynamic block pick-and-place algorithm.

The main lesson we learned was that it was vital to fully understand and make sure that everything is working at each stage before proceeding to the next, the stages being 1. simulation, 2. hardware, and 3. working against an adversary. And, in order to do so, it was necessary to start early. Because each part is cumulative and only adds more complications, it is important to be able to trust what has already been done, and to understand it fully so that changes can be made efficiently when they must be made. As has been said throughout the semester, if something does not work on simulation, then it will not work on hardware; and as we discovered, if something does work on an isolated arm in hardware, then it will not necessarily work in competition.

While our algorithm succeeded in that it brought us to second place in the class, it could benefit from further refinement to increase its robustness given the unpredictability we observed in the competition.

First, we only use closed-loop feedback when picking dynamic blocks, to verify if a block is in the grippers or not. We don't use it in static blocks because we use a deterministic assumption, but in competition it was observed that a static block's position could be falsely detected, resulting in the gripper picking up nothing. Incorporating tactile feedback into the static blocks would eliminate one possible fail-point.

Visual feedback could also be utilized more powerfully. We knew but did not account for, the possibility that opposing teams would sweep blocks off of the dynamic block platform. If for any reason there were no remaining dynamic blocks on the platform, and there were still blocks on the static platform for us to pick, then it would be optimal to return to the static platform to continue pick-and-place from there. However, our implementation currently will have the arm stuck in a loop trying to capture dynamic blocks when there are none, which we observed in the final round (shown in **Fig. 2B.**) We could use visual feedback to confirm whether there are indeed blocks remaining on the platform before attempting the dynamic block-picking procedure.

In the end, it is impossible to predict all of the possible ways that things can go wrong in a competition setting against an adversary block-picking agent. Additionally, the assumptions made of this competition environment (constant angular velocity of the dynamic platform and static obstacles otherwise) are not necessarily true of other relevant applications of block-picking. To improve the robustness of the block-picking algorithm, it may be worth it to pursue a more sophisticated dynamic block-picking algorithm depending on the predictability of the environment and the variety of trajectories a block can move in. Finally, if there are more complex, non-deterministic obstacles in the environment that prominently obstruct a significant portion of the robot's workspace, then it becomes much more valuable to construct path-planning algorithms. All of these are future steps that could be made to improve upon the block-picking methods described within this report.

## 6 Competition results

With the strategies for picking and stacking the static and the dynamic blocks, we were very pleased with the performance in the final competition where we came **2nd overall among all the teams that participated and successfully stacked 7 blocks (4 static + 3 dynamic)**. As mentioned previously, we identified the areas of improvement based on our performance and further improvement could be made in the future scope of work.