# MEAM 5200 Final Project- Team 12

Kevin Paulose, Tejendra Patel, Saurav Agrawal, Aditya Chennapragada

December 21, 2023

## 1 Introduction

In this final project, we developed a robust and reliable solution for acquiring and stacking static and dynamic blocks and our proposed algorithm was tested in simulation and evaluated in experimental setups. Moreover, we used some new strategies for Positional IK that improve the algorithm's performance, which greatly reduced the time cost and speed up the movement. This paper is organized as follows.

- Explained our design for the algorithm in the Method section
- Described the testing and validation process in the Evaluation Section
- Analyzed our strategy and made changes to our algorithm accordingly keeping the competition structure in mind in the Analysis section
- Included the lessons we learned from this project in the last section.

## 2 Method

### 2.1 Static Block Strategy

The algorithm for static block manipulation is meticulously designed, initiating with the robot's movement to optimally position its end-effector and consequently, the camera directly above the static blocks. This strategic placement facilitates a clear view of the blocks' April tags. The `block_detection()` function is then deployed, yielding the transformation matrices of the April tags in the camera's frame. A complementary filter with a high alpha value of 0.9 is utilized for multiple detections refining the pose data, which is instrumental in minimizing noise and enhancing the accuracy of the detected positions.

Subsequently, the block's pose is converted from the camera's perspective to the robot's base frame, which serves as the origin for DH parameters and forward and inverse kinematics' calculations. This transformation reveals the block's position in the robot's coordinate system, particularly in the $z$-direction, facilitating the movement of the end-effector to the precise location using inverse kinematics, with the `franka_IK_EE()` function playing a pivotal role.

Since the robot's seventh joint or the end-effector joint is fixed in our inverse kinematics, we had to carefully consider the block's orientation in the end-effector frame. The `detector.get_H_ee_camera()` function is employed to transform the block's rotation matrix to the end-effector frame. The `z_swap()` function aligns the block's $z$-axis with that of the end-effector frame, and additional rotations are calculated to align the $x$ and $y$ axes for an effective grasp. The calculated orientation is then applied to the end-effector, enabling it to grasp blocks in various orientations without encountering joint limit errors or requiring excessive rotation.

Waypoints are established for placement of static blocks on the designated side, leveraging the analytical inverse kinematics. The success of block capture is verified using `get_gripper_state()["position"]`, and the placing height is incrementally adjusted for each block. If a block is not successfully picked, the height remains unchanged, and the loop persists until all blocks are handled.

This procedure delineates the fundamental algorithm for the static block's pick-and-place routine, with exhaustive testing conducted in both simulation and hardware to troubleshoot and resolve issues related to orientation, height adjustment, and gripper state, among others. The detailed functions and solutions to encountered challenges are elaborated upon later in the report.

The following pseudocode represents the algorithm for static block manipulation, as implemented in the provided Python code.

**Algorithm 1** Static Block Manipulation

Position robot for optimal camera view of static blocks
**function** BLOCK_DETECTION
    Detect blocks using `detector.get_detections()`
    Refine poses with `complementary_filter` (alpha = 0.9, interation=7)
    Transform block poses to robot's base frame
    **return** transformed block poses
**end function**
**function** STATIC_PICK_PLACE(target_placing, block_pose_world)
    **for** each target_block in block_pose_world **do**
        Adjust orientation with `z_swap` and `detector.get_H_ee_camera()`
        Compute and apply IK with `franka_IK_EE`
        Pick and place block, verify with `arm.get_gripper_state()["position"]`
        Adjust placing height if block is picked
    **end for**
**end function**

## 2.2 Dynamic Block Strategy

### 2.2.1 Initial Algorithm (Not used in competition)

Initially, for picking the dynamic block rotating on the turning table, we decided to move the robot to a position above the turning table to gain a view of the dynamic blocks. Using the detection function, we obtained the velocity of the blocks. The linear and angular velocities of the detected blocks are computed to facilitate dynamic tracking of their movements. Within a loop iterating over the detected blocks, the current and previous positions of each block are found. Subsequently, the linear velocity is determined by evaluating the difference in positions and dividing it by the elapsed time between two consecutive captures. The resulting linear velocity vector is then transformed into the world coordinate system using the rotation matrix of the robot's end-effector frame. Simultaneously, the relative rotation matrix between the current and previous orientations of the block is calculated, enabling the extraction of the angular velocity vector through the computation of a skew-symmetric matrix. Similar to linear velocity, the angular velocity vector is transformed into the world coordinate system. The computed linear and angular velocities for all detected blocks are printed, providing insights into the dynamic behaviour of the objects.

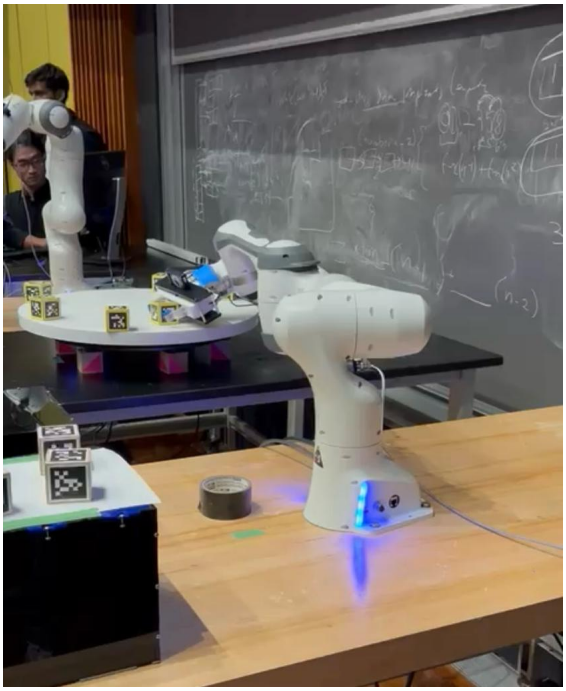
Following this, the code invokes the IK_velocity function, which performs inverse kinematics to determine the joint velocities necessary for the robot to align with the computed linear and angular velocities of the first detected block. This mechanism allows the robot to dynamically adjust its motion in response to observed changes in the positions and orientations of the detected blocks, showcasing a real-time tracking strategy.

But there were many problems that we faced with this such as the position of dynamic blocks were not fixed. Also deciding which block to pick first was not a trivial task and sometimes the block detected was moving in the frame of the camera so because of the time constraint we decided to not use this technique.

### 2.2.2 Dynamic Block Picking Strategy (Used in Competition)

Due to time constraints in hardware robot testing and issues with dynamic blocks in simulation (further details are discussed later in the report), we adopted a 'fishing' method for dynamic block picking. Through trial and error, we determined an effective robot orientation for sweeping from the outer edge to the middle of the turning table, as illustrated in the figure below. Our strategy involved the robot's end-effector sweeping towards the table center, pausing for 7 seconds to attempt gripping a block. If a block was successfully gripped, the robot would sweep remaining blocks off the table, hindering the opposing team's chances of picking dynamic blocks. This was followed by precise inverse kinematics to move to the placement position. If no block was detected, as indicated by `get_gripper_state()`, the robot would continue its task of picking dynamic blocks again.

(a) Oriented end-effector for Dynamic Block placing



(b) End-effector for Dynamic Block picking and sweeping

Figure 1: Lab 4 robot

As shown in the image, we adjusted joint 6 to prevent the robot from colliding with the table during sweeping. However, this angle presented a challenge when placing the block, as it held the block at an angle. To resolve this, we applied a corrective angle to joint 6 during placement, ensuring proper and centered placement of the block in the box.

Video demonstrating dynamic block picking, sweeping other blocks and returning to initial orientation- YouTube link

**Algorithm 2** Dynamic Block Stacking Strategy

Position robot for sweeping dynamic blocks on turning table.
**function** DYNAMIC_PICK
    Open gripper.
    Move to pre-calculated position near dynamic blocks.
    Sweep from table edge to center, pausing for 7 seconds.
    Attempt to grip a block.
    **if** block is gripped **then**
        Move to displace other blocks, hindering opponent.
        Adjust joint angles: $q[5] \leftarrow q[5] + \frac{\pi}{6}, q[0] \leftarrow q[0] + \frac{\pi}{5.5}, q[6] \leftarrow q[6] + \frac{\pi}{8}, q[4] \leftarrow q[4] + \frac{\pi}{18}$.
        Move to placement position using IK.
    **else**
        Continue sweeping.
    **end if**
    Adjust joint 6 to correct orientation for placing: $q[6] \leftarrow$ rectification angle.
**end function**
**function** DYNAMIC_PLACE(target_placing)
    Calculate placement orientation.
    Correct joint 6 angle for accurate placement.
    Place block in designated position.
**end function**

## 2.3 Final Strategy Combining Both Static and Dynamic Blocks

After thorough testing on hardware and calculating timings for each task, we devised a strategy for the pick-and-place of dynamic blocks. Our approach allowed us to stack four static and two dynamic blocks while simultaneously clearing all dynamic blocks from the turning table, preventing the opponent team from having any dynamic blocks to pick. The overall strategy began by detecting the static blocks and saving their positions and orientations. Initially, we picked the dynamic block first, sweeping our side of dynamic blocks from the turning table. Subsequently, we picked one of the static blocks. The rationale behind this sequence was that the dynamic blocks from the opposite side took time to reach our side. While picking the first static block, the dynamic blocks were now on our side. We repeated the dynamic block pick-and-place and sweeping the remaining blocks. Throughout this process, we continuously checked the gripper state to determine if it had gripped, incrementing the placing height accordingly.

During execution, we observed cases where 1-2 dynamic blocks were still remaining. To address this, we performed an additional sweep to ensure all dynamic blocks were cleared. We then proceeded to the static block pick-and-place phase. During this phase, we checked the gripper state, and if it missed any static block, the system would check for any remaining blocks and continue the loop if necessary, otherwise stopping.

This constituted our main strategy. Since all calculations were done from the robot's origin, it was straightforward to adapt the code for both robots by changing the sign of the y-axis.
Video demonstrating our complete strategy execution during competition- YouTube link

**Algorithm 3** Final Pick and Place Strategy

Initialize ArmController, ObjectDetector, and variables for Inverse Kinematics (IK).
Set robot to start position, display team information, and wait for user input.
Set arm speed and initialize variables for IK.
**function** BLOCK_DETECTION
    Initialize alpha.
    **repeat**[5]
        Get and transform block poses from detector to robot base frame.
    **until return** block poses in world coordinates.
**end function**
**function** STATIC_PICK_PLACE(target_placing, block_pose_world)
    **for** each block in block_pose_world **do**
        Adjust arm using IK, execute pick and place, and verify capture.
    **end for**
**end function**
**function** DYNAMIC_PICK
    Open gripper, move to dynamic block area, sweep, and attempt to grip a block.
    Adjust arm position and orientation for placement.
**end function**
**function** DYNAMIC_PLACE(target_placing)
    Correct orientation for accurate block placement and place dynamic block.
**end function**
**while** repeat flag is True **do**
    Move arm to initial target, retrieve camera transformation, and detect blocks.
    Execute dynamic pick and place, check gripper, and adjust height.
    Move to new target for static pick and place, execute for first detected blocks.
    Open gripper, move for dynamic block pick, and adjust arm for pick and place.
    Check gripper state, adjust arm, and move for block placement.
    Repeat arm adjustments and movements for block placement.
    Move to another target for static pick and place, execute for remaining blocks.
    **if** No remaining blocks **then**
        Set repeat flag to False.
    **end if**
**end while**

## 2.4 Functions in Detail

### 2.4.1 Analytical Inverse Kinematics - franka_IK_EE()

Initially, we used the code for Inverse Kinematics (IK) using a numerical method based on IK-Position. However, we observed that the numerical method was notably slow, taking approximately 10-15 seconds to converge, which was deemed insufficient for our application. To enhance the speed of the IK solver, we conducted further research and devised a strategy involving the fixation of the seventh joint of the robot, effectively transforming it into a 6 Degrees of Freedom (DOF) manipulator. Leveraging geometric principles, we then formulated an analytical Inverse Kinematics solution, which proved to be approximately 5 times faster than the numerical approach.

Our reference for this approach was the research paper titled Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design [1]. Delving into the details of the research paper, we extracted the necessary information related to the Denavit-Hartenberg (DH) parameters and Forward Kinematics that we had initially employed in our code. This process allowed us to align our implementation with the principles outlined in the research paper, resulting in a significantly faster and more efficient analytical IK solution for our 6 DOF manipulator.

The IK solution is obtained by employing geometric principles and formulas derived from the Denavit-Hartenberg (DH) parameters and forward kinematics.

#### Joint Equations

#### Joint 7 (q7)

Joint 7 is a revolute joint with a specified range. The code checks if the provided $q_7$ value is within acceptable limits. If valid, it assigns the value to the 7th joint of all four solutions; otherwise, it returns a 4x7 array filled with NaN.

#### Joint 4 (q4)

The code calculates Joint 4 using geometric relationships involving the end-effector pose and predefined parameters. It checks if the calculated value falls within the allowed range; otherwise, it returns a 4x7 array with NaN values.

#### Joint 6 (q6)

Joint 6 has two possible solutions, which are calculated based on geometric relationships involving the end-effector pose. The code handles angle wrapping and joint limit checks for both solutions.

#### Joints 1, 2, 3, and 5 (q1, q2, q3, q5)

These joints are determined based on geometric relationships, rotation matrices, and position vectors. Joint limits are checked for each of these joints, and if any joint angle is out of bounds, the corresponding solution is marked as NaN.

#### Multiple Solutions

The code returns a 4x7 array ($q_{\text{all}}$), where each row represents a solution, and columns correspond to joint angles. The four rows accommodate different possible solutions due to the nature of trigonometric functions and multiple configurations that can achieve the same end-effector pose. The NaN values in the array signify invalid solutions or configurations outside the joint limits.

We delved deeper into the process of obtaining the four solutions and carefully considered which solution best suited our specific task. Given the straightforward nature of our environment, characterized by minimal robot movement during static pick and place operations, we were able to pinpoint a single solution among the four. This selected solution consistently provided the Inverse Kinematics (IK) solution without encountering singularities.

Our observation led us to identify the elbow-down position as the optimal configuration for our scenario. By fixing this particular solution, we ensured a stable and safe orientation for the robot throughout the task. This strategic decision eliminated unnecessary instant configuration changes, contributing to both safety and

efficiency. The adoption of Analytical IK for the 7-Dof robot played a pivotal role in accomplishing the task with notable efficiency, requiring minimal computational power and time.

### 2.4.2 Complementary_filter() and Block_detection()

One of the major issue when shifting from simulation to hardware that we faced was that of detection of april tags. Because of the noise and lighting issue we were getting errors in pose detection. So to solve this we looked into different filters like median filter and complementary filter. We fianlly decided to use complemnetray filter as it was very fast with not much tuning where as median filter requied tuning parameter.

The complementary_filter function is defined to calculate a weighted combination of input data and reference data using a specified parameter, $\alpha$ (0.9). The block_Detection function employs this filter in a loop iterating five times. During each iteration, it captures block poses, applies the complementary filter to smooth the data, and subsequently transforms the poses into a world coordinate system. The final result, represented by the block_pose_world list, contains the processed world coordinates of the detected blocks.

In more detail, within each iteration of the loop, the algorithm collects block poses from a detector, applies the complementary filter to ensure a gradual transition between consecutive poses, and converts these poses into a NumPy array for computational efficiency. The loop then proceeds to the next iteration, repeating the process. Finally, after the loop completes, the algorithm transforms the filtered block poses into world coordinates using a matrix operation and returns the resulting block_pose_world list.

### 2.4.3 Orientation Function: z_swap()

While detecting static blocks using the AprilTag, we encountered variations in the transformation matrix provided by the get_detection() function. Specifically, the matrix was not consistently in the form of rotations around the x, y, and z axes ($R_x, R_y, R_z$). To address this inconsistency, we developed the z_swap() function.

This function serves a dual purpose: first, it ensures alignment of the z-axis of the block with that of the camera and consequently the end-effector. Second, it facilitates the adjustment of the rotation matrix to adhere to the desired format of $R_x, R_y, R_z$. By employing the z_swap() function, we harmonized the orientation data obtained from the AprilTag detection process, allowing for a more standardized and consistent representation of the static block's pose. This standardization was crucial for subsequent calculations and manipulations involving the detected block's orientation in our robotic system.

After this we got the orientation of block in camera which we transformed to end-effector frame as we were giving the angle to directly the joint 7. But here we noticed that sometime the angle was greater than $180^\circ$

**Algorithm 4** Swap Function

**function** SWAP($a, b$)
    $a, b \leftarrow b, a$ ▷ Swap values of $a$ and $b$
    **return** $a, b$
**end function**
**function** Z_SWAP($a, \varepsilon$)
    $column\_no \leftarrow 10$
    **for** $i \leftarrow 0$ **to** 2 **do**
        **if** $|||a[2,i]| - 1|| < \varepsilon$ **then**
            $column\_no \leftarrow i$
        **end if**
    **end for**
    **for** $i \leftarrow 0$ **to** 2 **do**
        SWAP($a[i,2], a[i, column\_no]$) ▷ Swap elements in columns
    **end for**
    **if** $a[0,0] \times a[1,1] < 0$ **then**
        **for** $i \leftarrow 0$ **to** 2 **do**
            SWAP($a[i,0], a[i,1]$) ▷ Swap elements in rows if needed
        **end for**
    **end if**
    **return** $a$
**end function**

### 2.4.4 Orientation Function: Orientation correction

Since in our algorithm we don't get the value of the last joint of the arm, we have to give that our own input. This function was to make sure that the end effector joint rotates *as least as possible* so as to avoid any values out of joint limits.

**Algorithm 5** Orientation Correction

**function** STATIC PICK PLACE($a, b$)
  **for** no. of blocks detected **do**
    R $\leftarrow T_{ce} * (T_{block})$
    $R_{3x3} \leftarrow R[:3,:3]$
    rz $\leftarrow arccos(R_{3x3}[0,0])$
    **if** rz < $\pi/4$ **then**
      rz $\leftarrow rz - \pi/2$
    **end if**
    **if** rz > $\pi/4$ **then**
      rz $\leftarrow rz + \pi/2$
    **end if**
    open gripper
    find target pos q with dummy q[6]
    **if** ($R_{3x3}[0,0] > 0$ and $R_{3x3}[0,1] > 0$)$or$($R_{3x3}[0,0] < 0$ and $R_{3x3}[0,1] > 0$) **then**
      q[6] $\leftarrow q[6] - rz$
    **end if**
    **if** ($R_{3x3}[0,0] > 0$ and $R_{3x3}[0,1] > 0$)$or$($R_{3x3}[0,0] < 0$ and $R_{3x3}[0,1] > 0$) **then**
      q[6] $\leftarrow q[6] + rz$
    **end if**
    go to target position q

### 2.4.5 Static_pick_place(target_placing, block_pose_world)

Combining all the functions above we were able to do the pick and place of static block accurately and fast. The primary function, static_pick_place(), orchestrates the pick-and-place sequence for each target block within the block_pose_world list. The core functionalities can be outlined as follows: Firstly, the orientation of the target block is meticulously adjusted using the z_swap function, The rotation matrix is further processed to compute the rotation angle, denoted as rz. In cases where rz surpasses a predetermined threshold, it undergoes adjustments to ensure it falls within a specified range.

The subsequent steps involve arm movement and gripper control. The position of the target block is transformed into an optimal picking location through inverse kinematics using franka_IK_EE function. The arm is subsequently guided to the calculated position and the gripper is closed (`arm.exec_gripper_cmd(0.049,50)`) to securely pick up the block.

An error-handling mechanism is incorporated into the code, verifying whether the gripper's position falls below a predefined threshold. If such a condition is met, indicating a potential failure in the picking process, the arm is maneuvered to a predefined recovery position and the height of placing is adjusted accordingly.

Following successful picking, the code proceeds to the block placement phase. The target placing position undergoes transformation via inverse kinematics, and the arm is navigated to the designated position. This process is designed for iterative execution, allowing the code to repeat the entire pick-and-place routine for multiple placement positions. By systematically adjusting the target placing position height, the robotic arm demonstrates a comprehensive and efficient approach to picking up static blocks from predefined locations and accurately placing them at specified target positions.

### 2.4.6 Dynamic_pick_place

This was a simpler algorithm, as we employed the fishing and sweeping method. It involved providing the IK_position with the previously tuned position and orientation of the end-effector. We began by setting the initial orientation of the end-effector just outside the turning table, as shown above in the algorithm section. Then, we moved only joint 1 to sweep it across the table. Although we checked whether it picked up the blocks,

adjustments in the height for placing the block were necessary. However, this proved to be a trivial task. More details about the position and sweeping angle can be found in code.

# 3 Evaluation

## 3.1 Simulation testing procedures

### 3.1.1 Static blocks

We have developed two different codes for red and blue robot stations, but this was easy for us because we only needed to flip one of the signs in all hard-coded start positions in our code.

Our strategy for picking static blocks starts by detecting the blocks using the function that was provided to us, and we picked them using our Analytical IK. Since we had to give the final joint our input, we had to check if the input we were giving was correct in terms of gripping the block perfectly.

We faced some problems with this strategy since we initially assumed that the z-axis of all the blocks would always be aligned with the z-axis of the gripper. Once we realized the inconsistency, we implemented the necessary changes in our algorithm to align the block and the gripper properly. Our strategy to check orientation was to make the gripper hover over the blocks as opposed to actually going and picking them up, so we don't waste time by restarting the simulation every time.

After streamlining orientation verification, the next step was to stack the blocks. We dropped the blocks from a small distance from the topmost block in the stack, so we tried to make sure that all the blocks were stacked perfectly without any errors to ensure that the stack was stable. Again, issues with the orientation caused us problems in picking the block (blocks were slipping from the gripper when not gripped correctly) or while placing the imperfectly picked blocks (which led to the formation of imperfect and unstable stacks). Since we had to give the final joint our input, we had to check if the input we were giving was correct in terms of gripping the block perfectly.

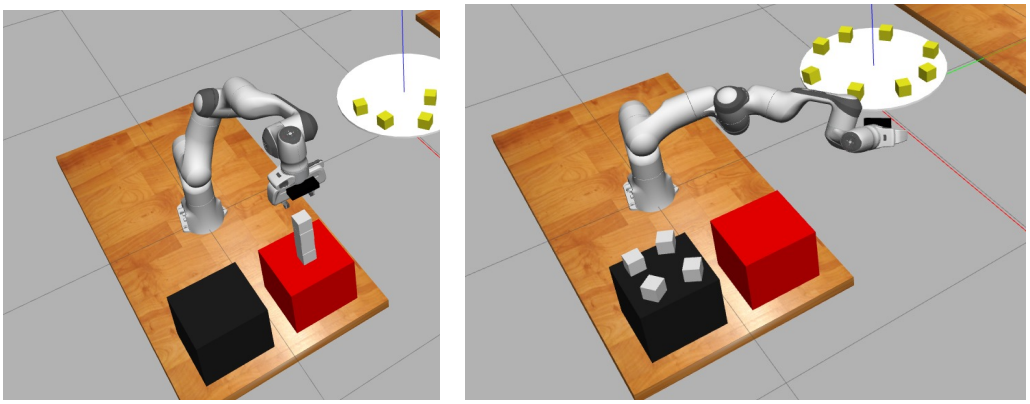
As a whole, we used the simulation to tune certain parameters like the placing configuration, the height from which each block is being placed, etc.

### 3.1.2 Dynamic blocks

Here, the transition between the red bot and the blue bot took slightly more time because we also had to obtain the correct joint configurations. To that end, we used IK_position_null which was developed in the lab to fine tune the positions by giving it various configurations and seeing which configuration worked best. Once we found a suitable start configuration, we hard-coded that configuration to save time.

We couldn't test our strategy for picking up dynamic blocks because our "disruptive" approach couldn't be simulated well in the environment that was provided. Since our strategy mainly involved waiting a certain time for a block to enter the gripper and then making a sweeping movement to topple other blocks off the turntable, the simulation environment couldn't understand the sweeping movement and the collision of the blocks with the end effector.

However, we did use the simulation environment to obtain configurations for our tactics. We have used the simulations to finetune our sweeping approach, by making changes in the sweeping arc, finding proper start and end configurations for the sweeping arc and fishing for blocks while avoiding collisions, which was critical. We took factors like collision with the table, correct angle of picking the block, correct angle of placing the block to avoid instabilities in the stack, etc.



(a) Stacking in simulation

(b) Dynamic block pick

Figure 2: snaps from simulation

## 3.2 Lab testing procedures (hardware)

### 3.2.1 Static blocks

Our testing strategy in the lab was the same as the strategy used in the simulation environment. We tested orientation and the pick place operations by hovering the end effector over the blocks as opposed to picking them up to save time.

We faced a lot of issues while working with the real robot like detection issues, orientation problems etc. To tackle the detection issues we faced, we had implemented a complementary filter so as to get least error in detection of blocks. We have detected each block seven times and passed all seven readings through the filter.

As for the orientation issues, we have figured out the problem by looking at the camera and how the coordinate system of each block is being defined. We also faced some issues with the np.arccos function since it only gives values from (0,pi). We also observed that past a certain rotation angle, the blocks were being picked erroneously, so we had to design a function to properly assign angle to the end effector by using both arccos and arcsin functions.

We also tuned various parameters like block placing height and placing configuration since there was always a certain error in the actuation of the end effector in its z-direction. To avoid collision, we had to tune exactly where the end effector is going to hold the block so we avoid collisions at any cost. This meant that we had to grip the block by its upper half, which sometimes led us to grip the block with lesser force. We therefore also tuned the force and position parameters of the exec_gripper_cmd() function so we have tight and foolproof gripping.

Static block stacking in lab- YouTube link

### 3.2.2 Dynamic blocks

We initially planned to read the velocities of each block to pick them, making sure there were no relative velocities between the end effector and blocks. The problem we faced in this approach is that IK_inverse was very slow and the success rate was very low ( 40%). We came up with a new solution which was also probabilistic but more reliable.

We tried to tune the start configuration of the sweep because blocks sometimes got stuck in between the robot arm and the turntable. This sometimes caused the computer to give out a collision alert, so we had to avoid that scenario to avoid disqualification. We had to drop dynamic blocks at a certain angle because they were being picked up at a certain angle. We had to change the dropping configuration since the angle meant that the block was being dropped with some error in the x-direction. We have tuned these parameters in such a way that the drop will not affect the stability of the stack.

After we made all of these corrections, we tried to stack the blocks multiple times so we could record the repeatability of our algorithm. Extensive testing was done on both red and blue robots.

Dynamic block pick and sweep in lab- YouTube link

#### 3.2.3 Full simulation run combining both

We knew that our strategy depended more on toppling the blocks off the table than stacking them. Once we figured our static, sweeping and dynamic strategies, we wanted to test the whole strategy together by using various iterations and combinations. We initially wanted to pick up a couple of dynamic blocks, sweep all remaining blocks and then start placing static blocks. We tried this, but this meant that the dynamic blocks were at the absolute bottom of the stack costing us points. Hence, we decided that we would place a dynamic block first, sweep the blocks on the table, stack a static block, go back to pick another dynamic block, sweep other blocks off the table, go a third time to make sure all blocks are swept off, and then stack all static blocks.

We also had to consider the case where a block has been missed while fishing for dynamic blocks, so we wrote an algorithm to check if a block has been picked by using the get_gripper_state() function. We were incrementing the dropping height after each block had been placed, so we had to check if the block was being placed, if not, we didn't increment the height. Most of our testing for the dynamic blocks and our sweeping strategy was tested on hardware because it couldn't be tested on simulations as extensively.

# 4 Analysis

## 4.1 Preliminary results

Our preliminary attempts had several setbacks because we faced a lot of problems with orientation and gripping that rooted from orientation faults. Once we corrected that, we were able to successfully stack all the static blocks in all attempts with both robots, and we were able to also sweep the table clean all the time, with the success in picking at least one dynamic block about 90% of the time and success in picking two dynamic blocks at about 75%.

Our strategy's first sweep is timed to happen at about 15 seconds after the code is run, and the second sweep happens about 40 seconds later. This was owing to us changing the speed of the bot itself using set_arm_speed().

Both the blue and the red robots were able to stack both static and dynamic blocks successfully according to our "disruptive" strategy. From our extensive simulation and lab testing described above, we determined that the robot behaved as expected. We saw that the robot grabbed one of the dynamic blocks and swept the area to remove the block from the table. Then, it placed a static block. By that time, the rest of the dynamic blocks came to our side, and we tried to grab the other block and swept the table again, devoiding the dynamic table of the blocks. We also did a 3rd sweep only to ensure all the blocks were in the singularity for the robot or of the table.

## 4.2 Analysis of optimal stack configuration

After extensive testing, we determined that the most optimal configuration that would score us the points in the competition was to stack one dynamic block, one static block, one dynamic block again and then three more static blocks. This configuration of the blocks would give us our maximum of 10500 points. Since we were using geometric analytical IK, our robot motions was already very fast and we were well within time. So, our secondary intent was to disrupt the other team's stacking.

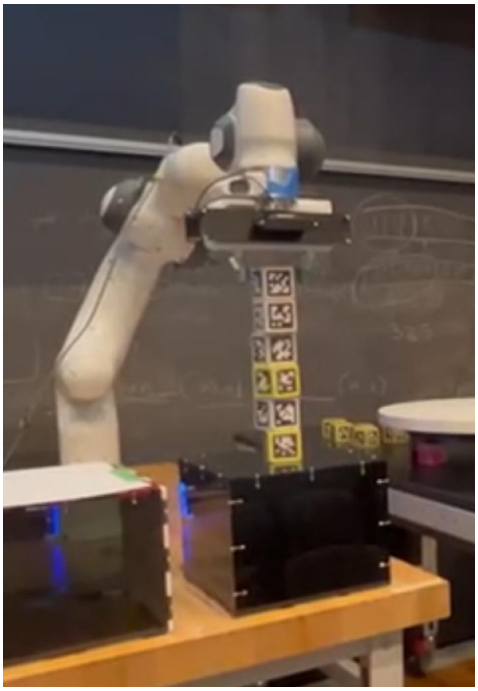


Figure 3: Dynamic, static, dynamic, static, static, static (bottom to top) configuration

## 4.3 Game day- 13 December, 2023

**Division 3**

For the first round in division, we scored 4000 points total on the red robot, clocking 2 minutes and 20 seconds (40 seconds left), stacking four static blocks correctly and sweeping all the dynamic blocks. In second round, we scored 1750 points on the blue robot, clocking 2 minutes and 6 seconds (54 seconds left) by placing two dynamic and three static blocks on the box (all independently without making a stack). The first static block did not place well and toppled rendering all other following blocks to topple off. This gave us 5750 points in total for the first round and crowned us the winner of Division 3.

Video of the robot on game day for Division 3 round 1- YouTube link
Video of the robot on game day for Division 3 round 2- YouTube link

We faced a problem with the gripper's force on match day. This was probably due to a misaligned gripping sponge on the end effector which made our force not being delivered as planned and it led to loose gripping. This was evident in our second match in division 3, where we were not able to create a proper stack because of misalignment and imbalance.

**Quarterfinal- Top 8**

For the first round in the quarterfinals, we scored 10,500 points by stacking six blocks in the order of one dynamic, one static, one dynamic and three static on the red robot, clocking 2 minutes and 15 seconds, while our opponent scored 500 points, making us the round winner.

Video of the robot on game day for Quarterfinal match- YouTube link

**Semifinals- Top 4**

For the semifinals, we scored 2500 total points on the red robot, and a collision was detected between the bot and the block, leading to our disqualification from the round. We were able to stack one dynamic and two static blocks and sweep all the dynamic blocks from the table, leaving the table empty for the opponent team. Our

opponents could stack all the static blocks, securing 4000 points, thus advancing to finals.

Video of the robot on game day for Semifinal match- YouTube link

**Playing for the podium**

For our last single elimination round, we scored 2500 points on the blue robot and we got disqualfied for the same reason the semifinals round because of collision detection. Our opponent scored 400 points on the red robot, by stacking all the static blocks losing the podium.

Video of the robot on game day for third place match- YouTube link

### 4.4 Introspect

Although we made it to the semifinals with ease, we were eliminated here because of us overlooking the increased speed input we gave to the bot. Due to this, the end effector did not rotate in the time it reached the target configuration, which led to a collision with the block. If we would have added a checkpoint on top of the target block before actually picking them up, we would've been safer although it would take us a couple of seconds more(we did have plenty of time to spare). We believe that we could've won the competition if only we had not overlooked this corner case because the teams that played the finals all went for the dynamic blocks after stacking the static blocks, by which time we would've rid the playing field off of all the dynamic blocks. This would have given us a two-block edge over our opponents, giving us the win by a sizable margin.

We could achieve a maximum of 10,500 points on the red bot (quarterfinal match) when everything went according to plan, and we could stack 6 blocks alternating dynamic blocks with static in the first 3 blocks, and then all static blocks. Our highest on the blue bot was a total of 2500 points when we were able to stack a dynamic block and 2 static blocks in that order during our 3rd place match.

## 5 Lessons Learned

From comprehending how manipulators operate and forming DH parameters to understanding forward kinematics at the beginning of this course, to finally executing pick-and-place tasks with a 7DOF robotic manipulator, the journey has been unforgettable. Butterflies fluttered in our stomachs on the day of the competition, wondering if our robot would work, and we felt a mix of anger and sadness when it collided with the last block that could have secured our victory. Throughout this entire experience, we gained valuable insights.

The most crucial lesson we learned is that transitioning from simulation to hardware is not a trivial task, and the inconsistency of hardware can pose challenges. Due to time constraints, we couldn't thoroughly test our algorithm on both robots, making it susceptible to errors that led to our failure in the final round. We mistakenly assumed the hardware was flawless, despite our perfect strategy and dynamic sweeping in every round. We strategically disrupted the opponent to prevent them from obtaining any dynamic blocks and successfully placed six blocks. However, we overlooked the fact that increasing the robot's speed could lead to drifting. Controlling the seventh joint independently resulted in a collision, teaching us the importance of considering all cases and identifying real-world errors through hardware, not just simulation. Additionally, we could have implemented trajectory optimization, such as minimum acceleration or minimum jerk trajectory using polynomials, to address issues when the robot moved at high speed and collided with blocks.

Ultimately, the most significant lesson was the importance of effective teamwork and coordination. This project involved addressing numerous exceptions, from ensuring proper block gripping to fine-tuning placing heights and devising winning strategies. Many minds coming together were necessary for tasks like brainstorming strategies, and during the competition, we enjoyed putting on a show. In the end, we believe that our hard work paid off, and we emerged victorious.

## References

[1] Yanhao He and Steven Liu. Analytical inverse kinematics for franka emika panda – a geometrical solver for 7-dof manipulators with unconventional design. In *2021 9th International Conference on Control, Mechatronics and Automation (ICCMA)*, pages 194–199, 2021.