MEAM 5200 : Pick and Place Challenge

Renu Reddy Kasala, Raima Sen, Charlie Drazba, Benjamin Abt

I. Introduction

The objective of the final project was to design a robust solution to enable the Franka Emika Panda arm to perform the task of Pick and Place in a fast and safe manner. While we primarily aimed to achieve as many points as we could, we also noted the importance of conducting safe trajectory planning of the manipulator arm to enable collision free movement in a busy environment. The following report presents a complete strategy and mathematical derivation for this task, along with a detailed analysis with different test cases that were conducted on simulation and on hardware.

II. Method

A. Strategy for the Competition

In order to achieve a high score, the goal was to maximize points according to : $Points = ValuexAltitude$, where the value for static blocks was 10 and that of dynamic blocks was 20. A few important factors influenced this objective such as : (a) Number of blocks stacked (b) Value of the block stacked (c) Optimal trajectory that minimizes time taken between different configurations (d) Stacked structure design We came up with several different strategies to incorporate the above factors, but taking into account the various design considerations, we adopted the following sequence of decisions to successfully achieve the task :

1) At the start of 3 minutes, the manipulator arm shall perceive the turntable from a “dynamic home” position and estimate the pose of the dynamic blocks. It should then attempt to grab a dynamic block. There are 3 such attempts made in the beginning, after which it breaks out of the dynamic block loop.
2) The manipulator arm should then move to a “static home” position directly above the table where the static blocks are placed and perceive the poses of these static blocks. The arm should then iteratively pick each static block and place it on the goal table with the correct orientation to avoid toppling. The loop breaks when all 4 static blocks are stacked.
3) Finally in the remaining time, the arm goes back to the “dynamic home” position and attempts to grab the remaining dynamic blocks.

On the day of the final competition, we reduced the attempt to grab dynamic blocks in the beginning to a single attempt since at the start of 3 minutes the turn table is undisturbed and the poses of dynamic blocks is the most deterministic. While we were almost always sure that we would be able to pick and stack all 4 static blocks, we decided to pick some of the dynamic blocks first since it had a higher value, although our dynamic block picking strategy was more time intensive. This is a tradeoff we considered while designing our strategy. We also initially tried to make a pyramid out of the picked blocks to maximize the points, but on further testing we noted that the robot gripper would sometimes collide with the previously placed blocks on the goal table, which would lead to a software stop during hardware testing.

B. Perception

The input to the perception module is a list of AprilTag tuples (tag name, tag pose) obtained directly from the camera sensor. Therefore, the tag pose (transformation matrix) is always in the camera frame. In order to utilize this information for the Pick and Place task, the following steps were adopted:

Given:

- H^{ee}_{camera} (obtained from detector.get_H_ee_camera())

To find:

- H^{base}_{block} (pose of the block in robot base frame)

Now, using forward kinematics, the transformation matrix H^{base}_{ee} can be obtained by using DH parameters.

$$H^{base}_{ee} = H^{base}_{1} H^{1}_{2} H^{2}_{3} H^{3}_{4} H^{4}_{5} H^{5}_{6} H^{6}_{ee} \quad (1)$$

$$H^{base}_{camera} = H^{base}_{ee} H^{ee}_{camera} \tag{2}$$

Using the H^{base}_{camera} and the pose obtained from camera's detector.get_detections() H^{camera}_{block}, we get

$$H^{base}_{block} = H^{base}_{camera} H^{camera}_{block} \tag{3}$$

Using the transformation of the blocks in Base frame, we then call the IK solver to reconfigure the robot pose to grab the dynamic/static blocks.

C. Inverse Kinematic Solver

After obtaining the blocks' pose data from the camera sensor, the manipulator arm should move towards the block in order to grab it. For this, we initially implemented an analytical solver based on [3]. However, this solution was limited to the assumption that joint 5 is unconstrained. Therefore we devised a gradient descent-based Inverse Kinematic solver that retrieved the desired joint angle configurations in an iterative manner, given the current and target homogenous transformations w.r.t the base frame. The time taken by the solver to arrive at a solution ranged from 0.5 to 11 seconds. The following algorithm was used:

Algorithm 1 Algorithm for IK

```
Input: target transformation, current joint configuration
Output: final joint configuration $q$
    Initialize $q = q_0$
    while error > tolerance && iter < maxiter do
        calculate error $e$ between current and target position $o_n^0(q) - o$
        calculate gradient $J^T e$ where J is the linear velocity Jacobian $J_v$
        calculate q step $\Delta q = -\alpha J^T e$
        update $q = q + \Delta q$
        if (error < tolerance && $q \in jointlimits$) then
            break
        end if
    end while
    return $q$
```

Hyperparameter Tuning : We primarily tuned 2 hyperparameters : learning rate (α) and convergence threshold (σ). Having a very low learning rate leads to slow convergence to the goal joint angles. A high learning rate may cause overshoot from the minima. So tuning the learning rate was very crucial. We tried different approaches with the gradient descent and after thorough analysis we used Step function-based gradient descent on IK, where if the convergence threshold is achieved (end effector is very close to the goal), the learning rate drops by 30%. This radically improved the convergence speed to the goal with 4 being the lowest number of iterations for convergence.

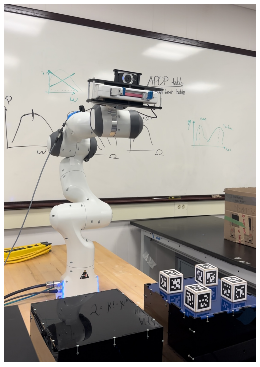


Fig. 1: Start position

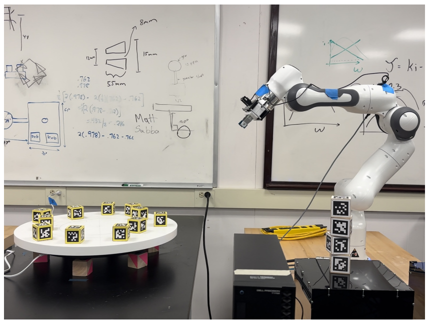


Fig. 2: Dynamic home position

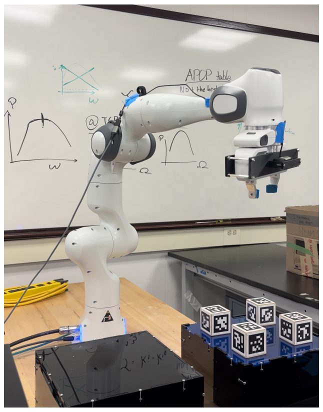


Fig. 3: Static home position

Jacobian : The Jacobian matrix of the Franka panda arm for a given configuration is a 6 x 7 matrix. We used only the linear velocity Jacobian (J_v) 3 x 7 matrix. This allowed us to accurately determine the final positional configuration to reach the blocks. This strategy proved to be sufficient for grasping static blocks but it was insufficient for dynamic blocks. The following section covers orientation filtering to align the end effector with the orientation of the block.

D. Orientation Filtering and Tracking

In order to align the robot end effector with the orientation of the static or dynamic block and grasp from the top, we calculated the euler angles after the IK solver achieved the target position. The following equations were used :

$$R_{block}^{base} = H_{block}^{base}[:3,:3] \tag{4}$$

It was noted that although the AprilTags were present on all faces of the block, the orientation of the block frame was such that the z axis of the block frame was not always facing upwards to the table. This was handled by simply reordering the columns of R_{block}^{base} as per Right-Hand Rule. Additionally, the end effector frame should be aligned but inverted by 180○ with the new z-axis of the block so that the grasping occurs vertically from top. In this way we obtain the filtered rotation matrix that the end effectorshould achieve. The following functional representation portrays this filtering :

$$(filtered)R_{block}^{base} = invertZ(reorder((raw)R_{block}^{base})) \tag{5}$$

The angles obtained from IK solver q_{target} are further passed through the Forward Kinematics algorithm to obtain $H_{position\_target}$.

$$H_{position\_target} = FK(q_{target}) \tag{6}$$

$$R_{position\_target} = H_{position\_target}[:3,:3] \tag{7}$$

Now we get the difference between $(filtered)R_{block}^{base}$ and $R_{position\_target}$ using the following equation :

$$R_{error} = (filtered)R_{block}^{base}R_{position\_target}^{T} \tag{8}$$

Using R_{error} we obtain the euler angles by using this algorithm:

Algorithm 2 Pseudocode for Euler angle calculation

```
Input: $R_{error}$ with elements $R_{11}, R_{12}$ ...
Output: euler angles $\psi, \theta, \phi$
  if ($R_{31} \neq \pm 1$) then
    $\theta_1 = -asin(R_{31})$
    $\theta_2 = \pi - \theta_1$
    $\psi_1 = atan2(R_{32}/cos\theta_1, R_{33}/cos\theta_1)$
    $\psi_2 = atan2(R_{32}/cos\theta_2, R_{33}/cos\theta_2)$
    $\phi_1 = atan2(R_{21}/cos\theta_1, R_{11}/cos\theta_1)$
    $\phi_2 = atan2(R_{21}/cos\theta_2, R_{11}/cos\theta_2)$
  else
    $\phi$ = anything; can set to 0
    if $R_{31} = -1$ then
      $\theta = \pi/2$
      $\psi = \phi + atan2(R_{12}, R_{13})$
    else
      $\theta = -\pi/2$
      $\psi = -\phi + atan2(-R_{12}, -R_{13})$
    end if
  end if
  return $\psi, \theta, \phi$
```

The euler angles obtained are used to change $q_1, q_2$and q_3 since the first 3 joint angles are responsible for the orientation of the end effector. Although we implemented this on the robot for orientation tracking, we noted that the end effector would still sometimes wrongly collide with the block or handle the block from corners, leading to an unstable grip. We were unable to debug this entirely, however, on some attempts, the orientation of the end effector would align in a perfect manner with the block.

E. Trajectory Planning

As an initial strategy, we were planning to use off-the-shelf algorithms like A-star or RRT. However, we noted that a simple strategy derived from Artificial Potential Fields would suffice for the given challenge. Since there was no obstacle or busy environment, we ignored the likelihood of the robot hitting nearby blocks. We also observed that methods like A-star although provided good trajectories, the solution generation time was extremely slow, which would have affected our competition run time. Once the IK solver and orientation filter modules were implemented, we used them sequentially along with applying Forward Kinematic on a few hard-coded strategic waypoints to conduct the entire trajectory planning for dynamic and static blocks. A detailed implementation strategy for both static and dynamic blocks is provided in section H and section I.

F. Gripping

Once we obtain the desired q_{goal} for static/dynamic block using the IK solver and orientation filtering, the next goal is to grasp the block. Using the following steps, a block is grasped :

1) We use **arm.exec_gripper_cmd(pos, force)** to open the gripper first. Here the **pos** is the distance between the 2 ends of the gripper which we set to 0.7m (any value greater than the width of the block which is 0.05m) and **force** is set to 0.
2) We then use **arm.safe_move_to_position(q)** to move the arm to q_{goal} configuration.
3) Then we use **arm.exec_gripper_cmd(pos, force)** to close the gripper to a width of 0.05m and force of 70N on the block.
4) After picking the block, the arm moves upwards to an intermediary waypoint q_{wp} between the source table and goal table. This is done again using the **arm.safe_move_to_position(q)** function.

G. Stacking

The stacking/dropping module consists of the following steps:

1) For each block to be stacked, we keep a track of a position at a safe distance above the pile from which we drop the block from. This is a running height.
2) From the intermediary waypoint q_{wp} the arm moves to this position using **arm.safe_move_to_position(q)** function. This is a running height that gets incrementally updated by the width of the block (0.05 meters).
3) We open the gripper to drop the block using the **arm.exec_gripper_cmd(pos, force)** function where gripper width is 0.7m and force is 0.
4) After stacking the current block, the arm continues picking blocks.

H. Dynamic Blocks

In our approach to handling the acquiring of dynamic blocks in the lab/competition, we used the below strategy:

1) From the start position, go to "Dynamic Home" position using **arm.safe_move_to_position(q)**. "Dynamic Home" is a set of joint angle values that were hard coded. This configuration acts as the observation configuration to view all dynamic blocks on the turn table from either the red and blue side.
start_position = [-0.01779206, -0.76012354, 0.01978261, -2.34205014, 0.02984053, 1.54119353+π/2, 0.75344866]
The following dynamic home observation configuration is when we are on the red side, dynamic_home = [π/2, -0.76012354, 0.01978261, -2.34205014, 0.02984053, 1.54119353+π/4, 0.75344866]
2) From this configuration, all the dynamic blocks on the turn table are visible. We estimate the velocity of the turn table using the following equations :
H^{ee}_{camera} (obtained from detector.get_H_ee_camera())

$$H^{base}_{ee} = H^{base}_{1} H^{1}_{2} H^{2}_{3} H^{3}_{4} H^{4}_{5} H^{5}_{6} H^{6}_{ee} \quad (9)$$

$$H^{base}_{camera} = H^{base}_{ee} H^{ee}_{camera} \quad (10)$$

After this, we obtain the first time stamp t_1 using Python time. At this time stamp t_1, H^{world}_{block} is calculated using the following equations,

$$H^{base}_{block} = H^{base}_{camera} H^{camera}_{block} \quad (11)$$

$$H^{world}_{block} = H^{world}_{base} H^{base}_{block} \quad (12)$$

From H^{world}_{block} we get the position data P_1 of the block at time stamp t_1. At a subsequent time stamp t_2 the same data is sampled. We can call this position data P_2. Using P_1, P_2 and $dt = t_2 - t_1$, we get the velocity of the turn table using the following set of equations :

$$r_{P_1} = \sqrt{x^2_{P_1} + y^2_{P_1}} \quad (13)$$

$$r_{P_2} = \sqrt{x_{P_2}^2 + y_{P_2}^2} \quad (14)$$

$$r = \frac{r_{P_1} + r_{P_2}}{2} \quad (15)$$

$$L = (x_{P_2} - x_{P_1})^2 + (y_{P_2} - y_{P_1})^2 \quad (16)$$

$$\theta = \arccos(\frac{2r^2 - L}{2r^2}) \quad (17)$$

To obtain a better estimate of the velocity, we collect θ for a number of blocks that are visible by the camera and take the average of θ. Finally, velocity is expressed as,

$$V = \frac{\theta}{dt} \quad (18)$$

3) After estimating the velocity of the turn table, we conducted several trials to check the time taken by IK solver to converge to a solution. This value varied from as low as 0.5 seconds to a maximum of 11 seconds. We additionally measured the time taken by the arm to move from the ”Dynamic Home” position to one of the dynamic blocks. This was roughly 5 seconds on the simulation. After multiple rounds of testing, we arrived at an upper bound of estimation time to be **16.7** seconds. This value was hard coded.
4) Now after having an estimate of the turn table velocity and trajectory time period, we open the gripper.
5) Based on all the observed blocks, we get $H^{world}_{block_{curr}}$
6) We predict the future pose of all the observed blocks (pose attained by different blocks after 16.7 seconds). For this, we use the following equations,

$$\theta = V dt \quad (19)$$

$$T_{table} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (20)$$

$$H^{world}_{block_{future}} = T_{table} H^{world}_{block_{curr}} \quad (21)$$

7) We obtain the $H^{world}_{block_{future}}$ now in base frame $H^{base}_{block_{future}}$
8) Among all the obtained future positions of different dynamic blocks, we find the block that is closest to the base of the robot by simple Euclidean distance between the base of the robot and the last column of $H^{base}_{block_{future}}$.
9) We then use the IK solver to achieve $H^{base}_{block_{future}}$ configuration from the ”Dynamic Home” position. This retrieved set of joint angles is q_{goal}
10) The gripping module from section F is called and the robot arm moves to an intermediary waypoint in order to avoid collision with the goal table.
11) Next, the stacking module from section G is called and the arm stacks the dynamic block and then goes back to the observation position (dynamic home position).

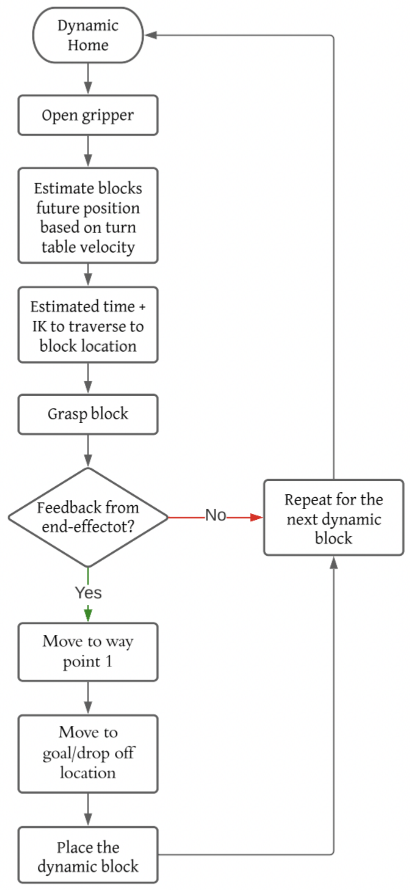

Dynamic Home
Open gripper
Estimate blocks future position based on turn table velocity
Estimated time + IK to traverse to block location
Grasp block
Feedback from end-effectot?
No
Repeat for the next dynamic block
Yes
Move to way point 1
Move to goal/drop off location
Place the dynamic block

Fig. 4: Dynamic block stacking strategy

I. Static Blocks

In our approach to handling the acquiring of static blocks in the lab/competition, we used the below strategy:

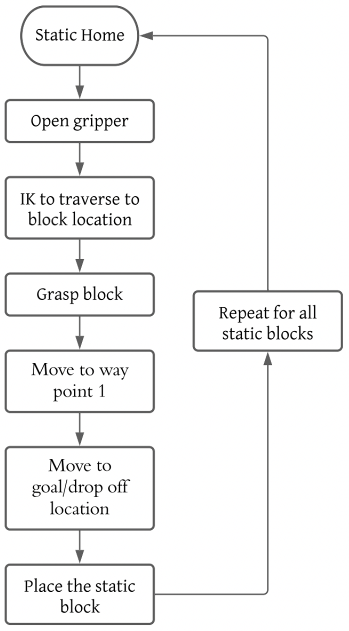
1) From the start position, go to "Static Home" position using **arm.safe_move_to_position(q)**. Similar to "Dynamic Home", "Static Home" is a set of joint angle values that were hard coded as as the observation configuration to view all static blocks on the source table from either the red and blue side.
   start_position = [-0.01779206, -0.76012354, 0.01978261, -2.34205014, 0.02984053, 1.54119353+π/2, 0.75344866]
   The following static home observation configuration is when we are on the red side, static_home = [-0.30, 0.05, 0.1, -1.5, 0.1, 1.5, 0.784]
2) From this configuration, all the static blocks on the turn table are visible.
3) We iterate over each of the observed static blocks one by one. Using H^{base}_{block} and current joint angles we use the IK solver to retrieve q_{goal}
4) It was observed that the IK solver converges faster for static blocks as compared to dynamic blocks.
5) The gripping module from section F is called and the robot arm moves to an intermediary waypoint in order to avoid collision with the goal table.
6) Next, the stacking module from section G is called and the arm stacks the static block with a running height and then goes back to the observation position (static home position).

Figure 5 shows a block diagram that portrays the strategy adopted for static blocks.

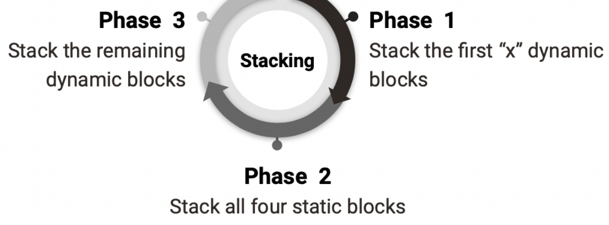
J. Code Structure

The code implements the above functionalities in a modular fashion

1) calculateFK.py is the Forward Kinematic module
2) position_ik.py is the gradient descent base Inverse Kinematics solver
3) calJacobian.py is the Jacobian matrix calcu lator
4) dynamic_block_estimation.py is the modul for detecting and grasping dynamic blocks
5) euler_angle_calc.py is orientation filtering module


Static Home
Open gripper
IK to traverse to block location
Grasp block
Move to way point 1
Move to goal/drop off location
Place the static block
Repeat for all static blocks
Fig. 5: Static block stacking strategy

In the code, we first initialize the required homogenous transformation matrices for both the red and blue sides. These are the H^{world}_{base}, H^{world}_{goal}, static_home, dynamic_home and intermediary waypoints for static and dynamic blocks.
Next, in the code we attempt to stack (x=3) dynamic blocks first, which takes about 45 seconds to run on hardware. After this in phase 2, all 4 static blocks are stacked iteratively. The time taken by phase 2 is about 65 seconds. Finally, for the remaining time, 20 attempts are made to grab the remaining dynamic blocks from the turn table and stack them on the goal table. A running height is maintained in the code to keep a track of the height of the end effector from the last block placed on the tower.


Phase 3
Stack the remaining dynamic blocks
Stacking
Phase 1
Stack the first "x" dynamic blocks
Phase 2
Stack all four static blocks
Fig. 6: Overall stacking Strategy

III. Evaluation

The evaluation process was divided into 3 broad stages :

A. Analytical Evaluation

1) Inverse Kinematics:

a) Analytical Inverse Kinematics: As stated previously, the analytical inverse kinematics approach was first attempted as per [3]. To evaluate the correctness of this approach, random joint angles that were within the joint limits were generated, and then placed into our forward kinematics (FK) solver. The output of this FK solver (H_{ee}^{base}) was then placed into the analytical IK solver. The solver we had initially implemented from [3] explained 2 versions. One was a generalized solver which would provide a joint angle set of 6 solutions each time. The second version proved to be more relevant to our evaluation, which was a case-consistent IK solver. The case-consistent IK solver generated the same set of joint angles for the given 2 parameters - Target transformation and current joint angles. While this implementation was straightforward and consistently gave the correct solutions for joints 4,5,6,7, the overall results we obtained from the analytical approach were subpar, due to some potential bug that we were unable to debug. After many days spent on testing it was determined that a pivot to gradient IK was necessary.

b) Evaluation of the gradient descent based IK: We tried different approaches with the gradient descent:

- Constant learning rate approach:
  Testing was done via generating randomized joint angles (within joint limits) and generating the FK position and orientation. This was then plugged back into the IK solver and results were compared to that of the original joint angles. Joint angles that were returned which were not the same as the original ones were plugged into FK, to determine if the combinations could yield the correct end effector position and orientation. This approach did not work well because the learning was either too large that it bounced off the solution or it was too small the it took long time to converge. In order to converge quickly without overshooting we experimented with adaptive learning rate.
- Adaptive learning rate approach:
  We tuned the learning rate such that it takes larger steps initially and then takes smaller steps as it gets closer to the global optimum. But this approach led local minima (which is not optimal). Testing was done similarly to that done in the constant learning rate approach.
- Learning rate based on Step function:
  Since there was no obstacle in the environment, we used a large learning to converge quickly, but when the error did not seem to reduce below a set threshold, it indicated that the gripper was close to the goal transformation. This is when we would set the learning rate to a lower value (decrease of 30%). This approach worked fairly well with our model. Again, the testing of the correctness of returned values is the same as that mentioned in the constant learning rate approach.

TABLE I: Learning Rate Comparison

Learning Rate	Iterations to converge
Constant LR	>500
Adaptive LR	60-100
Step LR	4-84

Parameters tuned : learning rate

B. Simulation Testing

We quantized the simulation tests to smaller subtasks.

Orientation Filtering :
The motivation behind this sub-task is to get a stable grip on the block being picked. This is possible when the end effector orients itself according to each block it plans to pick. In this way, the gripper always grasps the block on two opposite flat surfaces ensuring a firm grip.
In simulation, we extensively tested our solution for this sub-task for different orientations of the blocks. We ran multiple instances of our simulation for both Team Red and Team Blue but our solution passed only on cases when the block faces were slightly parallel to the table. In environments where the blocks were largely oriented about the table's z-axis, the gripper would either have an unstable grip as it would grip the block from the

corners. On other occasions, the IK solver would take longer to converge, therefore increasing the computation time. However this filtering method performed fairly well for static blocks in most testing environments we saw on the simulation. For dynamic blocks, the performance of this filtering method was worse due to the probabilisti orientation of the dynamic blocks.

Correct location on Rewards Table :
After having gripped a block in the rigl orientation, we moved on to the task of placin it in a suitable location on the reward table. A stable base ensures a stable stack. Hence, it was important to stack each block at the same location and in the same orientation. We tested this multiple times in simulation. We tuned the (a) drop height and (b) stack location. We also did multiple runs to hard code the best initial seed value for our IK Solver, to avoid it failing to find a path from the intermediary waypoints to the reward table.
While our solution worked well in simulation, we realized that the drop height and stack location didn't perfectly translate onto the hardware. After tuning those parameters to work on the Franka Panda arm in the lab, our solution worked every time.

Dynamic Block Stacking:
We ran multiple instances of our simulation for both Team Red and Team Blue for dynamic block stacking. During this evaluation we noted that the time to converge to an IK solution for the solver and the time taken to move from the dynamic home position to one of the dynamic blocks was roughly **16.7** seconds. Additionally, it was during the testing of the dynamic blocks on simulation that we were able to fine-tune the **"dynamic home"** position which was a crucial first step in order to observe all the dynamic blocks. The initial tests we ran did not have a dynamic **intermediary way point** that the robot arm must traverse through to place the block on the rewards table. This would sometimes lead to the block colliding with the rewards table and eventually falling off the grip of the end effector. In order to curb this, we introduced an intermediate waypoint. Although this increased the stacking time by a small percent, we had to make this trade-off in order to ensure that if a dynamic block was gripped, the end effector or the environment do not interfere with the gripped block.

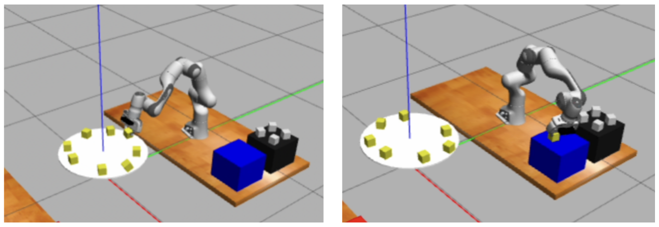


Fig. 7: Simulation testing: Dynamic block stacking on blue side

Static Block Stacking :
Similar to dynamic block simulation testing, we ran multiple instances of our simulation for both Team Red and Team Blue for static block stacking. We noticed here that the time taken to pick a static block was roughly 10 seconds when the block was placed parallel to the edges of the table. However, this convergence and picking time would increase to 45 seconds whenever the orientation of the block was highly rotated about the table's z-axis. This was potentially due to a bug in our orientation filtering method. In the simulation all the static blocks were picked in all the different environments we tested them in. Other parameters that we fine-tuned through static block simulation testing were : **"static home"** position, **running height** to place the static block on the tower, static **intermediate waypoint**, and correct **location on the rewards table.**

C. Hardware Testing

Dynamic Block Stacking :
When testing on hardware in the lab, we first confirmed the velocity of the turn table by timing one rotation of the table around the center. The estimated velocity was 3.6 degrees/sec, which was almost a match with the one estimated through the camera sensor (3.68 degrees/sec). However, we were unable to successfully grasp the dynamic block due to a repetitive error where the gripper was horizontally offset by a constant value from the dynamic blocks close to the circumference of the turn table. This error seemed resolvable and

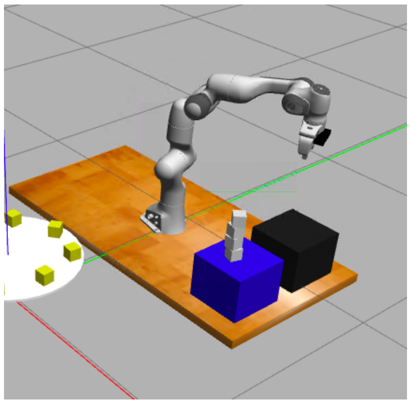


Fig. 8: Simulation testing: Static block stacking : Blue team

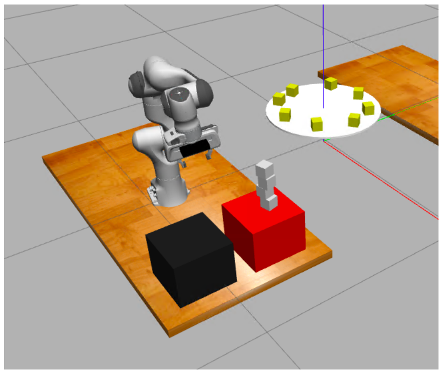


Fig. 9: Simulation testing: Static block stacking : Red team

no such error was observed during simulation. The robot arm would also very efficiently calculate the IK solution and converge to a dynamic block in the estimated time of 16.7 seconds. However, we could not resolve this constant error due to a lack of time and the unavailability of sufficient robot lab timings before the competition.

Static Block Stacking :
When testing on hardware in the lab, we were able to grasp the static blocks almost every time, despite the orientation bug. Our observation is that it was mostly because of the frictional gripper material that worked to our advantage. The running height estimate on simulation was also tuned for the hardware. The IK solver was surprisingly way faster on hardware than on simulation. We were able to stack all 4 static blocks in under 60 seconds.

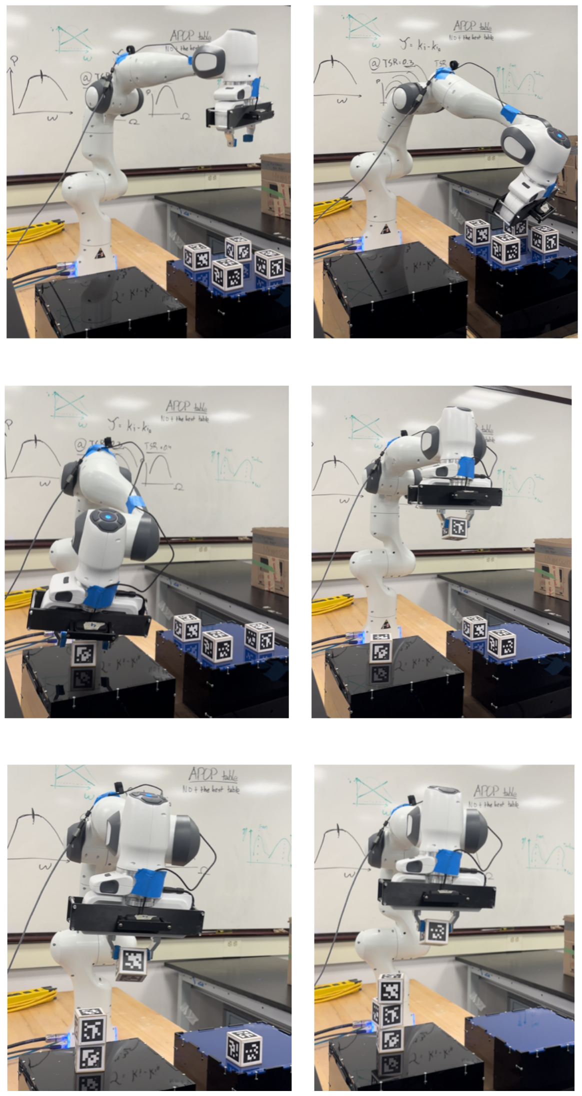


Fig. 10: Hardware testing: Static block stacking on blue side

IV. Analysis

Collision-free path planning is crucial given that the robot is supposed to operate in an environment that is not entirely static, i.e., it has dynamic obstacles. Although the expected environment is not going to change very quickly it is essential to plan the path in order to ensure that the interactions between the robot and the surroundings are completely collision-free and safe. Initially, we considered using a Potential Field planner or other off-the-shelf methods like RRT and A-star. While RRT method solved for the paths fast, it did not always yield in optimal paths. A-star on the other

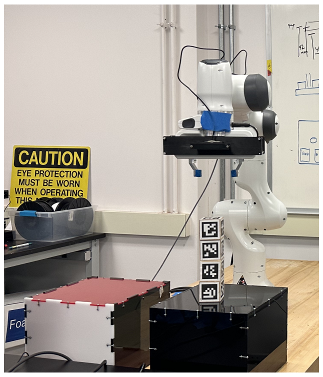


Fig. 11: Hardware testing: Static block stacked on red side

hand, generated optimal trajectories but the time taken to solve the trajectory was high. Since we were timed for 3 minutes, it was crucial to have a fast and accurate IK solver which led us to adopt a gradient-based solver with modifications.

A. *IK Static Collision*

From all of our testing and continuous tuning, it is clear that the end effector orientation was able to stack blocks consistently. Although we were satisfied with this metric, an analysis on the orientation of the IK orientation is necessary. With respect to static blocks, the IK solver had a preference for blocks 'more square' with the surface of the table, meaning that the sides were parallel with the surface of the table. Although this was the case, we were still able to pick up blocks not in this alignment due to the fact that the pressure of the closing end effector helped to actively rotate the blocks into an orientation that made them easy to grasp. Therefore, this proved to be a non-issue.

A bigger issue that we faced was the fact that the IK struggled to orient the end effector so that it's Z-axis was pointing along the Z-axis of the base frame. Generally, we found that the end effector tilted slightly forward, meaning that its Z-axis was pointing back towards itself. Examples of this can be seen in Fig. 7 and Fig. 10. In some testing situations, when the static blocks were placed close to each other and robot was going to pick up a block closer to itself, there would be a collision between the camera and the static block which sat behind the block the end effector was going to grab. During our second round of the final competition, this occurred, and we were compelled to go through a hardware stop.

Via close analysis of the code, we believe there were two reasons for such issues. First, in the calculation of the error between the current position and orientation versus the goal position and orientation of the end effector, there was no consideration for orientation. Secondly, in the gradient incremental step, we used a linear velocity jacobian, but there was no accountability for the angular jacobian. Essentially, our IK solver only converged to the correct position each time. Although this model was intentional, it was not optimal for picking dynamic blocks that were always changing their orientations.

To fully eliminate the issue of self collision with close static blocks, a strategy of going for the blocks furthest away from the end effector, then going for those closest was considered. Therefore, the tilt of the end effector had no chance of collision, as the blocks that previously caused collision were to be picked up first. Although this solution was decided on after competition, it was found that we could consistently pick up the static blocks without collision.

B. *Missing Dynamic Block*

While working with dynamic blocks, we found that sometimes the end effector was in the incorrect position radially, to where we wanted it. This was one of the few errors that we found when moving from software to hardware. This was most likely due to compounding errors from several sources in the dynamic block-stacking pipeline. The first source of error could be from inaccurately estimating the turntable velocity. We doubt that this is the case as we evaluated the efficacy of this method on the hardware via timing the turntable by hand, and comparing the results to that calculated.

Therefore, we think that the error was associated with determining the poses of the dynamic blocks which when combined with the uncertainty in the time it actually takes the arm to reach the predicted destination, caused the end effector to miss the block. The assumption is that the arm will reach the target destination at exactly the predetermined time in the future, but there is variability in the time it takes the arm to move. To account for this, the uncertainty in the time it takes IK to run was accounted for by waiting for a time that was longer than the known upper bound on the IK run time.

C. *Overall Competition Performance*

As a team, we are satisfied with our competition performance but know we could have done better. Had corrections to the dynamic blocks been done earlier, few more hardware testing hours been available, and had we strategized which stack blocks to grab first, we believe that we could have gone further in the competition. Nonetheless, we were able to stack both dynamic and static blocks and are happy with that metric.

V. Insights gained from this project

- This final project enabled us put all the modules(Forward kinematics, Inverse kinematics, Velocity IK and FK, Potential fields planning) learnt throughout the semester into practice. It was amazing to see how all the models fit together in a coherent matter.
- This project justifies what was mentioned way earlier in the lab, that is:
  - A system that does not work in simulation will not work in hardware.
  - A system that worked in simulation may not work in hardware.
- It was fun and informational to see the strategies implemented by different teams. We got to know how different teams focused on different areas - one focusing on sweeping the blocks off the turn table, while one focused to stack the dynamic first in the hope of grabbing them before their opponents, while the rest focused to implement the safe-static block stacking approach.

VI. Results and Discussion

Evaluation Criteria:
<u>Time < 3 mins</u>:
The time taken to pick and stack the 4 static blocks should be less than 3 minutes. This is because the competition runs for 3 minutes and stacking static blocks should be done well within that time.
<u>Number of static blocks stacked</u>:
The number of static blocks the robot is able to stack. Ideally, it should be able to stack all of the the 4 static blocks.

TABLE II: Results of Static and Dynamic Block Stacking

Trial environment	Number of trials	% of static cubes stacked	% of dynamic cubes stacked
Simulation	50	100	80
Lab	2	90	0

TABLE III: Final Competition Performance

Match	Number of static blocks stacked	Number of dynamic blocks stacked	Score	Time
1	2	0	1000	3 mins
2	1	0	250	3 mins

<u>Number of dynamic block stacked</u>:
The number of dynamic blocks the robot arm is able to pick.
Table 2 details the success rate we achieved in stacking dynamic and static blocks in both simulation environment and on hardware. Table 3 shows the final rollout of our performance on the day of the final competition. On the first match, we lost 45 seconds to grasp dynamic blocks in the first phase of our strategy of grasping 3 dynamic blocks. We still managed to stack 3 static blocks after that, therefore indicating our IK solver was quite fast. For the second match, we reduced the attempt to grasp dynamic blocks to just 1. The remaining time was devoted to grabbing static blocks. Unfortunately, the camera collided with one of the static blocks which compelled us to undergo a software stop. We debugged this issue after the competition by planning the trajectory such that the robot first grasps the static blocks closer to the robot operator and then proceeds to grasps the blocks behind them.

Links to videos : Link 1 Link 2

References

[1] Hutchinson, Seth A. and Mathukumalli Vidyasagar. “Robot modeling and control / Mark W. Spong, Seth Hutchinson, M. Vidyasagar.” (2006).

[2] Siciliano, Bruno and Sciavicco, Lorenzo and Luigi, Villani and Oriolo, Giuseppe. (2011). Robotics: Modelling, Planning and Control.

[3] He, Yanhao Liu, Steven. (2021). Analytical Inverse Kinematics for Franka Emika Panda - a Geometrical Solver for 7-DOF Manipulators with Unconventional Design. 194-199. 10.1109/ICCMA54375.2021.9646185.

[4] arXiv:2202.07869v3 [cs.RO] 29 Aug 2022