# Introduction to Robotics - Final Project Report

Introduction to Robotics (MEAM-5200)
Penn Engineering, University of Pennsylvania

*Aditya Jayant Ganapathiraju*
*Saptarshi Sadhu*
*Dheeraj Bhurewar*

---

# 1 Introduction:

This report presents our implementation of the MEAM-5200, Introduction to robotics class final project of the Pick and Place Challenge using the FRANKA EMIKA Panda robot arm. It also describes our testing strategy and the results of our testing on simulation and hardware followed by an in-depth analysis and comparison of the theoretical expectation from the code vs. the actual simulation and hardware results.

Two opposing teams are given three minutes with a possible two-minute overtime extension to stack as many dynamic and static blocks before the clock runs out. The primary focus is the dynamic blocks which are worth more points and time was taken to complete one block cycle as time is of the essence. The implementation of the block detection function which was used to locate the blocks in the view field of the camera on the arm was done by the teaching team.

# 2 Methodology:

We are using Inverse Kinematics (with gradient descent) for our approach to this challenge.

The methodology can be broken down into a few key steps as discussed in this section. The knowledge of the physical environment has been used where ever possible and has been crucial in planning the strategy for the pick-and-place challenge.

## 2.1 Block Detection

Block detection which is definitely one of the important aspects of the challenge was provided as a ready-to-use output from the camera along with the homogeneous transformation matrix of the camera frame w.r.t the end effector frame. Mathematically the location and orientation of the cubes detected by the camera can be described as:

block position = T0e @ H(cam) @ pose

After running the code the first time we printed out the H(cam) we stored the matrix in the $block - pos$ function.

$$\text{H(cam)=}\begin{bmatrix} 0 & -1. & 0, & 0.05 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -0.06 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Here, T0e represents the end effector's (EE) homogeneous transformation (HT) matrix with respect to the world frame, Hcam represents the camera's frame w.r.t the EE frame pose id the cube's detected HT matrix.

## 2.2 Inverse Kinematics

We employed a gradient decent approach where we set linear tolerance at $10^{-2}$m, angular tolerance at $10^{-3}$radians, maximum step to converge as 500, and minimum step size as $10^{-5}$. We found the IK solver was converging in 35-45 iterations.

## 2.3 Dynamic blocks

To pick up dynamic blocks the team's strategy was to go in blind and try to grab blocks that might be in the way and place them at predetermined locations. For the challenge, we attempted to pick dynamic blocks first and then move to the static blocks. The reasoning for this has been discussed later.
The dynamic block pickup strategy involves first, going to the $dynamic - sweep - start - position$ which we arrived at empirically, and then using $arm.safe-move-to-position()$ to go to $dynamic-sweep-end-position$ with an open gripper (like a jaw to grab blocks), this movement is diametrically coincident to the turntable and offers a high probability of getting blocks in its gripper, especially during the start of the game when the other teams are mostly focusing on static blocks. The probability of grabbing blocks decreases as we remove blocks from the turntable. Thus we decided to limit our number of dynamic block pickup attempts to two and then move to the static blocks. From our experimentation, we found the robot was always grabbing at least one block in the first two attempts.

## 2.4 Static blocks

The strategy with static blocks is to first read all positions of the cube, from the home position that we arrived at manually/empirically (from where we could read all four cubes in the simulation), convert them to world frame positions, solve IK for the cube positions, grab the block, go to an intermediate mid way location (to avoid collision with un-stacked blocks)and finally drop it at predetermined stacking positions.

## 2.5 Stacking

Our strategy was to make two separate stacks for dynamic and static blocks. the stack locations were selected by us and the homogeneous transformation matrices for the EE at each of these location locations were coded in. Every time when the program runs the IK is solved for these locations first before doing any task. This can be avoided by storing the IK solutions but we found that the homogeneous transformation matrices were easy to manipulate and only added

minimal computation time (2-3 seconds).
According to our stacking strategy, the arm attempts to stack two dynamic blocks first and then stack four static blocks, and then return to stacking up to 3 more dynamic blocks.

eg. HT matrix for the first dynamic block = $\begin{bmatrix} 0 & -1 & 0 & 0.5 \\ -1 & 0 & 0 & -0.2 \\ 0 & 0 & -1 & 0.23 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

# 3 Code Structure:

Here we have discussed how our game strategy was put into code. We have implemented the code structure in three distinct structures, the general inputs and data section, static block perception and planning, and finally dynamic block interface and planning.

Firstly we import a few functions which are critical in implementing this code, like the required Forward and Inverse Kinematics functions along with some fundamental math functions and robot interfaces for the proper execution of the robot code.

The $block - pos$ function is used to get the live current position of the scanned blocks and the joint position of the robot. This in turn helps us get the transformation matrix from the block to the world frame.

The $dynamic - blue$ function has a set of inputs for the blue team like the Dynamic start position which is the home position of the robot when targeting dynamic blocks, the dynamic sweep start position which is the position at which the robot is situated just before it starts to move forward (sweep) for dynamic blocks, and the dynamic sweep end position which is the position assumed by the robot once the robot has grasped the dynamic block, before it returns to the dynamic home position and then proceeds to the final home position, from where it will continue to place/stack the block. The commands to open and close the gripper ($arm.exec - gripper - cmd$ (0.1,50) for open and (0.04,30) for close respectively) are given in this loop accordingly. A check where $grip - dist < 0.025$, which is the clearance between the two pincers of the gripper, tells us if the block is actually grasped or not. If grasped it proceeds as mentioned above and if not grasped it resets to the dynamic home position before going through the dynamic block instructions again.

The $dynamic - red$ function is also similar to the $dynamic - blue$ function except that it has been engineered for the red team, almost a mirror of all the inputs w.r.t to the x-axis .

Next up, are the environmental parameters, which vary from team blue to red. Hence if the team is blue, the code goes through the blue team pipeline, from line 143 in the final.py file. Firstly a set of target locations for the different dynamic and static stacks have been given here in a certain sequence in order to account for the increase in height as the stack increases, along with certain offsets and tolerances considered for optimal stacking. A pose has been added here as well ($q - mid$) which is the intermediate pose for static blocks before they are placed in the target block.

| Parameter | Description | Location Array |
|---|---|---|
| start_position | Home position while scanning the static blocks | [0.18,-0.2, 0.02, -1.95, 0.1, 0.210+pi/2, 0.72] |
| dynamic_start_position | Home Position for the dynamic blocks | [pi/2-pi/20, 0.5, 0.02, -1, 0.1, 0.35+pi/2, -0.25*pi] |
| dynamic_sweep_start_position | The start position for the sweep process to grab dynamic block | [pi/2-pi/20, 0.32, 0.2, -2.17, -2.89, 0.77*pi, -pi+pi/5.2] |
| dynamic_sweep_end_position | The end position after the sweep process | [pi/2, 1.2, 0.2, -0.65,-2.89, 0.99*pi, -0.9*pi] |
| arm.exec_gripper_cmd(0.1, 50) | The function for gripper open | - |
| arm.exec_gripper_cmd(0.04, 30) | The function for gripper closed | - |
| arm.exec_gripper_cmd(0.1, 30) | The function for releasing grasped block | - |
| target_b_d(n) | The Target location of the ”nth” dynamic block in the stack | - |
| target(n) | The Target location of the ”nth” static block in the stack | - |
| q_mid | The Intermediate location assigned to static blocks | - |

**Table 1:** Comprehensive List of parameters used in code.

```
def dynamic_blue(q):
    # q is the state of the robot at the target stacking pos.
    dynamic_sweep_end_position = np.array([-pi/1.8, 1.2,  0.2, -0.65, -2.89,
                                           0.99*pi, -0.9*pi])
    dynamic_sweep_start_position = np.array([-pi/1.8, 0.32,  0.2, -2.17, -2.89,
                                           0.77*pi, -pi+pi/5.2])
    dynamic_start_position = np.array([-pi/2-pi/9, 0.5,  0.02, -1, 0.1, 0.35+pi/2,
                                           -0.25*pi])

    counter=0
    arm.safe_move_to_position(dynamic_start_position)  # dynamic home
    arm.exec_gripper_cmd(0.1,50)     #open gripper
    arm.safe_move_to_position(dynamic_sweep_start_position) # starts sweeping blocks f
    arm.safe_move_to_position(dynamic_sweep_end_position)   # end sweep from here
    arm.exec_gripper_cmd(0.04, 30)
    gripper=arm.get_gripper_state()
    grip_dist=gripper["position"][0]+gripper["position"][1]

    if grip_dist < 0.025:                                   #check if block is grabbed
        arm.safe_move_to_position(dynamic_start_position)   #if not go back to start
    else:
        arm.safe_move_to_position(dynamic_start_position)   #if grabbed, move here
        arm.safe_move_to_position(q)                        #move to stacking pos
        arm.exec_gripper_cmd(0.1, 30)                       #release block
        counter=1  ## if block is grabbed increment counter by 1

    return counter
```

**Figure 1:** The code pipeline for dynamic blocks for the blue team.

From Line 229 of the final.py code, The actual code pipeline begins, where the robot goes to the home position, and then to the dynamic home position and goes through the dynamic

pipeline, and attempts to pick 2 dynamic blocks before moving back to static blocks and going to the static block pipeline. If it successfully stacks the static blocks, it comes back to the dynamic blocks and attempts to stack some more before the time runs out. But if it still identifies any remaining static blocks, it will attempt to stack all static blocks until there are no static blocks left. It keeps track of the number of blocks actually stacked and the number of attempts and keeps track of the counter for dynamic and static blocks. The number of initial attempts for dynamic blocks can be changed using the function ($dynamic - i < 6$) which tells the robot to attempt to grasp up to 5 dynamic blocks before moving on to static blocks. (we provided stacking positions for 5 dynamic blocks only)

The same run-down with a few mirror and constraint changes is also implemented for the red team, which can be seen from line 396 till the end.

# 4 Evaluation:

The Strategy we have decided to undertake is to first go for the dynamic blocks to maximize the points in the competition, and after a certain amount of attempts or some critical time, the dynamic block attempts are abandoned and the robot prioritizes static block stacking. If any time is left after static blocks, the robot returns to dynamic blocks and tries to stack them before the time runs out.

Firstly, after the code is started to be implemented, the robot will go to the overall home position. From here it will go to the hard-coded dynamic home position and scan for blocks. Then it will perform "an intricate and tantalizing dance" of sorts to move to the dynamic sweep start position, move forward into the turntable, grasp the block and go all the way to the hard-coded dynamic sweep end position. The robot then will return to the dynamic home position, where it goes through a check to see if it has successfully grasped the block or not. If it has not grasped the block, and the check has failed, it will revert back to the original loop and proceed again to the dynamic sweep start position to proceed.

If the block is indeed grasped and the check is passed, it will move to the dynamic target positions, which vary according to the number of blocks and time remaining. To be on the side of caution, we have encoded two separate stacks for dynamic and static blocks instead of one mega stack if there is a collision, there is a chance to lose points as the stack might fall down or the blocks might fall off the target cube. After placing the first dynamic block, it will move to the dynamic start position and repeat the pipeline for dynamic blocks.

After the said number of dynamic block attempts have been completed, the robot abandons the dynamic block for the time being and targets the static blocks. It goes to the static block home position where it proceeds to scan the four blocks, before planning how and which static block to grasp. After it goes forward and grasps the block, it goes through a check to see if it actually did pick up a block. If it fails the check, it goes back to the static home position and repeats the process.

If the block is indeed grasped, it moves to an intermediate position, between the static

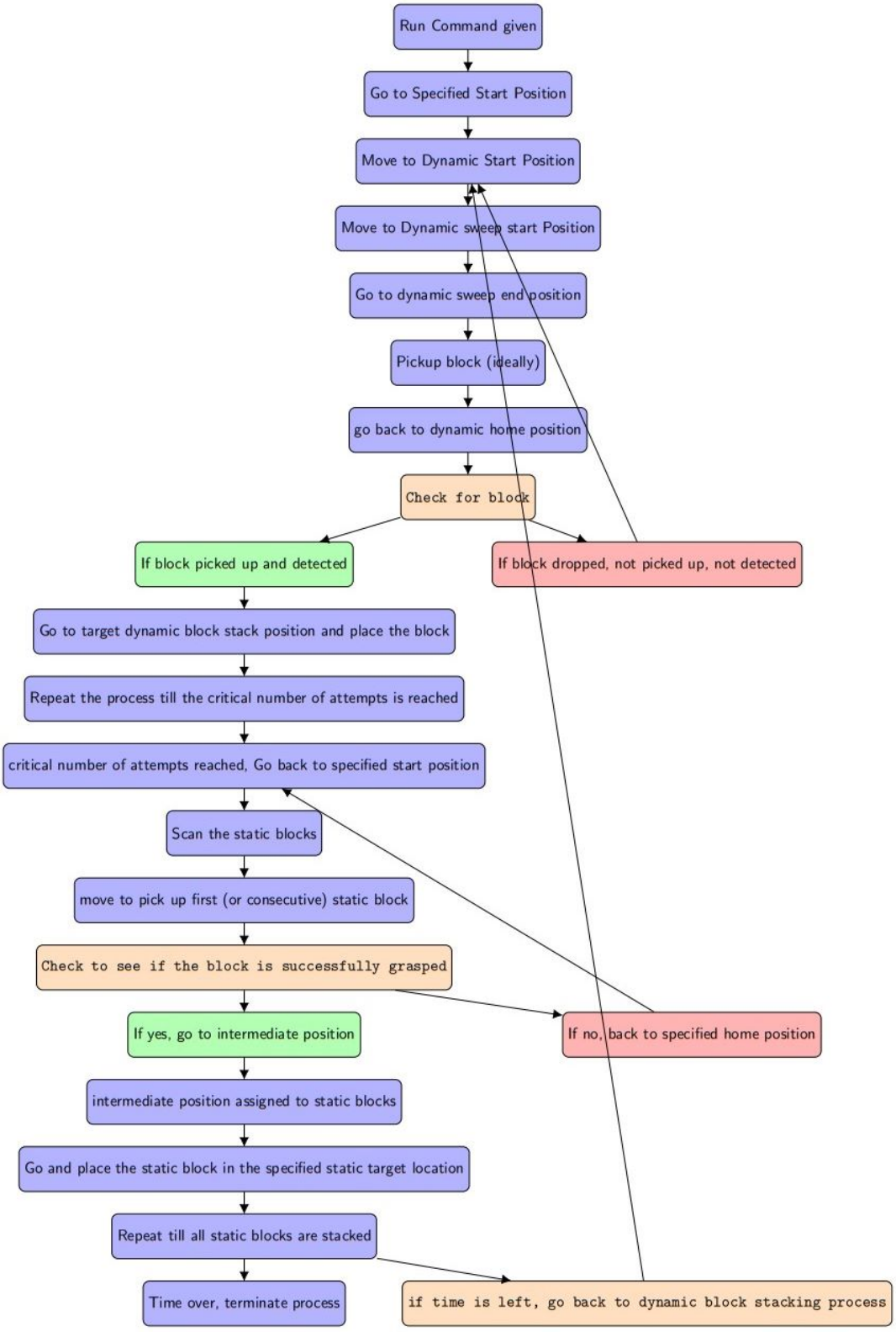


**Figure 2:** Flowchart of the Code Logic

block cube and the target block (q-mid), and then proceeds to the static block target position in the target cube and places it there. It then proceeds to go back to the static home position and repeats the static block strategy. The target block locations, with the necessary height corrections according to the block number have been defined in the code. After all static blocks have been stacked, it resumes or focuses again on the dynamic blocks and proceeds to return to the dynamic blocks and attempts to stack any number of them before the total time of 3 minutes runs out.

## 4.1 Testing (Simulation and Hardware (Lab)):

The first-ever hardware test for our test was in the actual competition itself, we, unfortunately, could not conduct any trial runs in the hardware prior to that. However, we did extensive testing in the simulation. We were consistently able to stack the static blocks and get the dynamic home position to attempt to place or stack the dynamic blocks in the target position. Note that our strategy involved sweeping the dynamic blocks and the blocks in the sim was too ”heavy” and the robot arm would roll over the simulated blocks but we knew that the motion was good enough o be successful in real life.

We have compiled all our results and documented them meticulously in a Google drive, which link is given here for reference.

Hardware Testing and Simulation Documentation Link

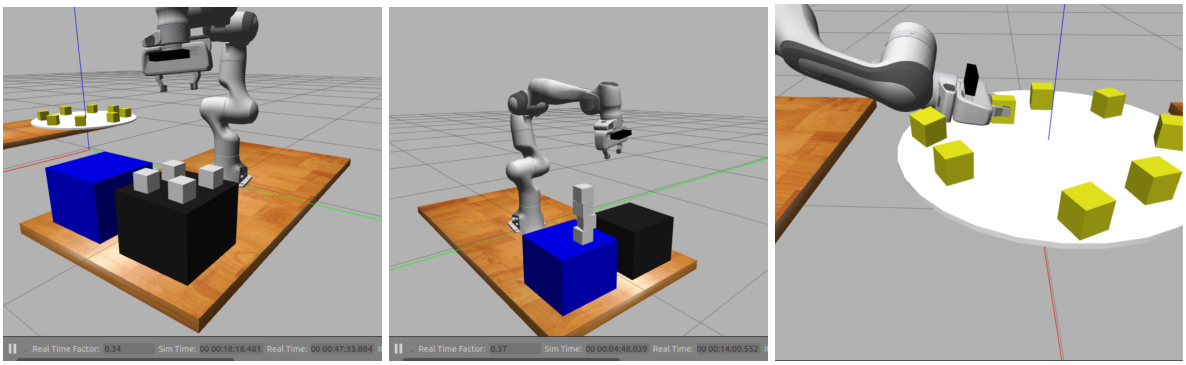

**Figure 3:** a. home position to scan static blocks b. a stacked tower of static blocks c. sweep in progress to grab dynamic block

- Pick and Place Challenge competition: During our first attempt (blue team), we managed to pick up the first dynamic block and come back to the dynamic start position before the gripper faultily opened and dropped the dynamic block. In our second attempt, we managed to grab the dynamic block, hold on to it, come back to the dynamic home position, and move to the final home position, but the robot dropped the block over the static block box. As the given time constraint for dynamic attempts was over, we tried to stack the static blocks for some points, but however, with the blocks in a non-standard

orientation, the block kept slipping out of the gripper, again and again, an astounding 7 times before we ran out of time. The worst of misfortunes.

In our second attempt (Red team), our robot failed to grasp the dynamic block in the first attempt. Whilst going into position to grasp the dynamic block in the second attempt, the TA manning the software stop used it, as the end effector scraped or touched another dynamic block whilst getting into position, hence ending our chance of stacking any blocks in this attempt. Again, no fundamental error in our approach, just plain bad luck.

- Day one of hardware testing: Our code worked in the competition, but it has issues with perception and grasping. A comprehensive check for errors, the error statement from the terminal, and simulation testing revealed that the 5th joint was out of limits and it was automatically constraining in simulation but this is not the case in actual hardware. We changed the limit from the original $-\pi$ to -2.89. This fixed the grasping issue, but it is still dropping the dynamic blocks over the static blocks and not placing them in the target black box. After checking the code, it works in simulation, and after some brainstorming later, the error was found to be in the gripper open function and was fixed.

  Now the robot is successfully able to place one dynamic and static block in the target space. However, it is grasping the same (the static block which is already placed in the target location) block, again and again, placing it in the target block in a loop. A hotfix would be changing the home block scanning location, such that only the un-stacked static blocks will be identified by the robot, not the already stacked static blocks. Another issue identified here, and confirmed by the TA is that the sensor values for force are faulty, and hence the gripper is dropping the blocks, this was later fixed.

  On the third try, two dynamic blocks from 3 tries, were successfully stacked along with one other static block. There was a collision risk while going for the second static block, so a software stop was initiated.

  In the final try for the day, we were successfully able to stack 3 dynamic blocks, and just place a few static blocks, however, an index error was identified, an error in the dynamic stack position for red and blue teams, which was later fixed.

- Day two of Hardware testing: The results were much more promising and successful this day, both in terms of accuracy and probability. On the first try, the robot grabbed one dynamic block and placed it successfully, and missed the second attempt as there were too many blocks clumped in one spot. In three attempts, two static blocks were placed (not stacked) and one block kept slipping repeatedly (a repeat of the scenario in the actual competition). A fix was implemented that if the static block is slipping, then the gripper is to change its orientation by rotating 25 degrees, before attempting to grasp the block again. On the second try, three dynamic blocks were successfully stacked and all four static blocks were placed (not stacked) in the target blocks, our highest ever.

  The final try was implemented after modifying the dynamic start, dynamic sweep start, and end positions for better grasping and performance against dynamic blocks. It was pushed a little forward to enhance the reaction time of the robot. The same result as in the previous paragraph, but in lesser time, leaving more time to attempt more dynamic blocks and implement more fixes.

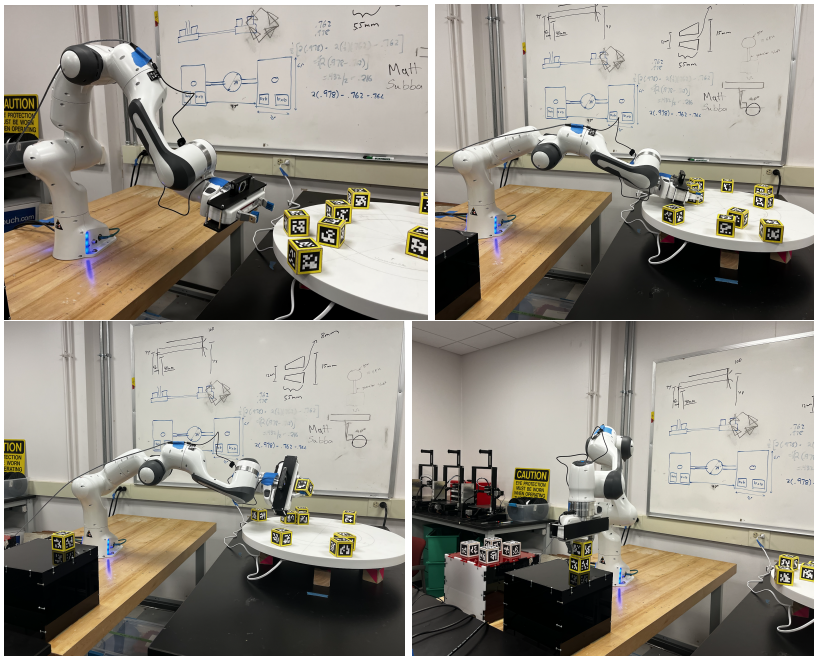

**Figure 4:** a.Dynamic sweep start position b. Sweep end position reaching the unreachable space! c. Picking up d. Stacking dynamic block

## 4.2 How often does my code succeed and how long does it take to run?

Currently, every 3 in 5 dynamic blocks are successfully grasped, picked up, and placed in the target block. We estimate the misses to be caused by tricky orientations of the block, and clusters of blocks coming up, hence not allowing one block to be picked up. It takes about one minute and fifteen seconds for the robot to stack two dynamic blocks. However, one very impressive feat achieved only by our team is that we are able to pick up dynamic blocks from the so-called unreachable workspace in the turntable and the TA has confirmed this.

The static part of the stacking challenge is being completed very quickly and with great efficiency and precision. It takes about 1 minute and 30 seconds to stack all static blocks and it almost always grasps and places the static block in the target locations.

## 4.3 Repeatability:

The repeatability of this challenge is fairly decent, most people who have completed Lab 1 and Lab 2 of the class should be able to get the robot going, albeit with a few glitches or errors. The first and most challenging aspect at the start would be figuring out the Inverse Kinematics of the robot as it plays a very important role in this challenge. Then comes the actual setup of the virtual environment in ROS and Gazebo, while it is straightforward, the setup can be buggy and a bit finicky. The simulation results vary very much from the actual hardware test as the blocks and their dynamics such as friction, resistance and weight are very different. Hence it is fairly challenging to replicate or implement for a beginner.

### 4.4 What would happen if the environment changed or if my implementation changed?

While the Implementation can change from team to team, like the change in block or stacking strategies, home, start, end, and intermediate positions, the Inverse kinematics used and its dynamics, the real-life changes in the environment, etc. It can be said to be fairly adaptable to changes in implementation.

Changes in the environment, however, can be tricky to adapt to, at least in some cases. The speed of the turntable, the orientation, distance, and placement of the static and dynamic block cubes and target cubes. The placement of the robot can be determinant as well, as it might change the reachable workspace of the robot, hence requiring changes in strategies. If there is a significant change in altitude from object to object or cube to cube, then there are some changes to be introduced as well.

## 5 Analysis:

For ease of analyzing the entire project we have further bifurcated the entire lab into 2 sections, namely:

- **Pre-stacking:** Includes all the analysis prior to picking the blocks (Block picking up methods used, etc.)
- **Stacking:** Includes all the analysis of picking and stacking the blocks.

Below we discuss each of the above-listed sections in detail.

1. **Pre-stacking:**
   Initially while deciding on which methods we should use for picking up the blocks we were torn between potential fields, IK - Analytical and IK - Gradient Descent.

   The *potential field* method is a very simple and computationally light algorithm. Also, it is flexible and can incorporate various components of a problem and reacts to changes in the environment at every step, which is why we first thought of implementing it. It is not so perfect algorithm owing to its major disadvantage of getting trapped at the local minima. Thus when we tried implementing it, we were skeptical if it will work robustly, primarily for dynamic blocks. More so because the blocks are constantly changing their positions. Implementing it meant that our algorithm should be robust enough to not get stuck in local minima while the blocks were moving. Thus, we started to look for other methods to pick up the blocks rather than through *potential field* even though we were able to get decent results while picking up the static blocks. Through our analysis, we have come to the conclusion that *potential field* is best used in cases where the environment has more stationary objects than dynamic ones for easier implementation.

   Similar to *potential field*, the *IK (Analytical method)* is easy to implement. But we were not able to figure out which of the possible solutions should our algorithm select in order

to get to the right position. Sometimes it did take the right solution, while sometimes it did not. We were not able to make our implementation robust which is why we tried implementing *gradient descent*. Though initially, we were not able to make the *gradient descent* converge in a lesser number of iterations, it was the method that fetched us with the best results, both static and dynamic. Thus, we went ahead with it and just tried to better our *gradient descent* implementation

2. **Stacking:** The main questions that we pondered upon post getting to the block were:

   - How do we make our Panda arm to avoid collisions?
   - What if the arm fails to grab a block?
   - How do we orient the gripper?

   So, thinking on those lines we figured that having an intermediate position somewhere between the initial and target blocks helped us avoid collisions with the blocks. We decided an intermediate position for both the dynamics and static blocks in order to avoid collisions on both ends.

   Now, to solve the problem of the gripper not grabbing the block we asked the arm to go back to the scan position and scan again to see if it now detects one block lesser than what it was detecting before. If it does see one block less then it has grabbed it, if not then it has failed it grabbing the block. But implementing the above technique was not so easy as there were reflections of the block from the shiny surface below so it would detect that as well and assume the reflection to be a legit block. Although there are lot of ways to solve the reflection problem, one of them being to check for orthogonality, we realized that this whole pipeline is not the most optimal way of solving the problem of 'failure to grab a block'.

   So, we then just looked for helper functions and tried to implement it to solve this problem. We first implemented the *get-gripper-state* function and tried to extract the force value making the assumption that if the arm did grab the block it should return some finite value and it does not grab the block it will return nothing. But while testing we observed that the *get-gripper-state* function did not return any values if it did grab the block. Thus, we tried extracting the position of the gripper instead through *get-gripper-state*. After testing this we observed that it was returning faulty values and it was corrupting the entire algorithm. After multiple iterations, we added a tolerance and that mitigated the problem.

   Now, the reason for the gripper to miss a block was that it was trying to grab it diagonally. This happened at times when the block was positioned at an angle to the gripper and was not parallel to the gripper. So, we thought of ways of aligning the gripper but could not implement anything very robust. One thing we tried was to rotate the orientation of the gripper with respect to its previous orientation by some angle until it grabs the block but that did take up a lot of computing time in the worst of cases.

   Another challenge was to make the dynamic stacking faster. Initially, we thought of

grabbing the dynamic blocks by measuring the speed of the turntable and then grabbing it from the top. The implementation for this took up a lot of time since we had to compute the velocity and then also wait for a block to arrive in the camera vision. Also, we were not able to grab blocks oriented by a slight degree sometimes (as discussed above) which is why we thought of coming up with another strategy. We decided to grab the block from the sides instead of grabbing it from the top because not only the algorithm was faster in grabbing the block (did not spend time calculating the speed of the turntable) but it also solved the problem of grabbing blocks with tilted orientations with respect to the gripper.

The last thing to make our algorithm faster was to manipulate the joint velocities. We did realize during testing that this would've made our implementation faster but we were not able to implement it. Given more time with the lab, we would have worked on it to make our algorithm run faster.

## 5.1 Challenges

- One of our first challenges was to get the environment up and running. Once we did that we struggled to arrive at a good solution to make the robot arm reach the blocks. To solve this problem we tried out hand in analytical IK and then moved to IK with gradient descent, which gave us good solutions.
- We found the simulation had some deviations from the real world. (9cm shift on the y-axis) which was corrected later. The simulation was much slower than the real arm motions and we later concluded that more dynamic blocks could have been attempted before the static blocks(more than 3). It took 1 min 2 sec to attempt 2.
- In the real world, the static blocks were slipping when the gripper tried to hold them diagonally. whereas in the simulation a diagonal grip wasn't an issue.
- The get-gripper-state() function did not work properly and only returned correct gripper states occasionally. this caused a dynamic block to be released during the competition
- The time was a great constraint and one way to place more blocks would have been to use velocity control.
- The dynamic block pickup strategy we employed worked really well for the initial 2-3 blocks (during our hardware tests) but for more blocks, the probability of grabbing a block decreases as the number of available blocks decreases and we are essentially going in blind without scanning the table.

# 6 Acknowledgements

We would like to thank Prof Cynthia Sung for her constant help support and guidance in this course and beyond. We are also grateful and thankful for all the Teaching staff, especially Erica, Archit, Aditya, and Wei Hsi.