Pick and place in static and dynamic environments using a 7 DOF Franka Panda arm

Anirudh Kailaje*, Dhruv Parikh*, Parth Sanghavi*, and Pranav Gunreddy*

*Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania

December 23, 2022

Abstract

Pick and place robotic arms are rapidly changing the dynamics found in the workplace. This is mainly because they are replacing human hands in handling repetitive tasks and other operations that don't require human intervention. While these arms find relevance in numerous applications and tasks, pick and place operation forms the core of it and hence is of utmost importance. This report focuses on how our team approached the bi-annual competition Pick and place challenge hosted by the Dept of MEAM, the University of Pennsylvania, as part of the course MEAM 5200: Introduction of Robotics. The scope of this letter is to implement and assess different planning and control methods and come up with a solution which is a complete module with minimal hard coding (dependencies on the environment). We start with a brief introduction to position and velocity control and types of planning methods and then explain the dynamics of the competition, the strategies we adopted, and what could have been done better.

Keywords
Inverse Kinematics, A*, RRT*, Potential field planning, Bezier curves, Jacobian, Dynamic planning

1 Introduction

Pick and place robots are used for a wide range of applications in the industry, starting from the assembly, sorting, inspection, and packing. They also are an integral part of robotics in Medical & surgery, food, and defense applications. Hence there is an abundance of work in this field that is relevant to the competition.
[1] elaborates on the control modes available, from position control to velocity control, and the advantages and complexities involved. Endpoint trajectories have been generated with each control mode, and metrics for performance assessment have been discussed. Solving Inverse kinematics has been quite a challenge throughout. [2] gives a detailed explanation of the geometric method to solve using kinematic decoupling. [3], [4] also provide interesting insights into solving the IK problem. We have referred to [5] & [6] to explore the state-of-the-art path and motion planning techniques that would give optimal trajectories in a dynamic environment. [7] was also a paper that had intriguing ideas about generating minimum jerk trajectories, which we tried to implement. Finally, we also looked at previous year's project reports of the competition [8], [9], [10], which had amazing strategies and approaches

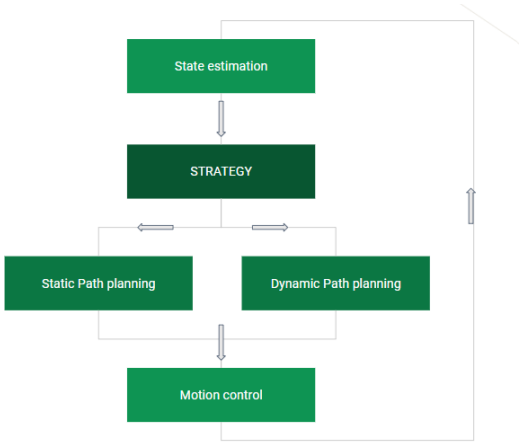


Figure 1: Methodology

2 Methodology

The following section describes the methodology that was used in the final competition. Keeping in mind the adversities of hardware setup, we made sure we designed our code such that it not only works well in simulation but also handles any anomalies in real testing autonomously. Our code as described in 1 is primarily divided into 4 sections -

1. State Estimation
2. Path Planning
3. Motion Control
4. Static/Dynamic Pipeline

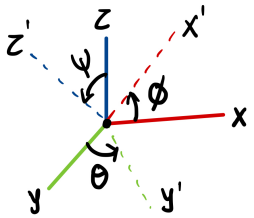


Figure 2: Roll Pitch Yaw representation

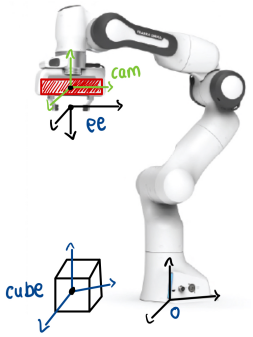


Figure 3: Camera Setup

2.1 State Estimation Pipeline

State Estimation is of utmost importance for any control application. We required a safe execution of updating all robot states such that it can make the main code look cleaner and the user doesn't have to worry about initialization. Our state estimation pipeline estimates 10 robot states, 4 static and 8 dynamic block frames w.r.t the robot base with high fidelity. Our state estimation module is broken into two subsections - Perception and Robot States.

2.1.1 Robot States

Robot State estimation is a straightforward implementation from previous labs. However, the functions have been made "safe" such that they are safe from erroneous variables. We estimate the End effector (x,y,z) in meters using forward kinematics. We also calculate End effector velocity $(\dot{x}, \dot{y}, \dot{z})$ using forward kinematics and then differentiating it using the discrete differentiation.

$$f'(x) = \frac{f(x + \Delta t) - f(x)}{\Delta t} \tag{1}$$

Here $\Delta t = t_2 - t_1$ is calculated when the function to update the states is called. These are the lateral States of the robot. For the angular states, we need the angle of rotation of the end effector around the X, Y, and Z axis. We can get this from the transformation matrix if we know the Euler angle convention used (it can be XYZ, ZYZ, or any other). To avoid this confusion, angles were calculated using a dot product with a base frame (inertial). In Figure 2, (x,y,z) represents Base Frame, and (x',y',z') represents the end effector frames. From this, we can calculate angles using equation 2:

$$\begin{aligned} \phi &= cos^{-1}(x \cdot x') \\ \theta &= cos^{-1}(y \cdot y') \\ \psi &= cos^{-1}(z \cdot z') \end{aligned} \tag{2}$$

2.1.2 Perception

The perception module was the system's primary source of noise and had to be dealt with methodically. Emphasis was laid on getting the block frames noise-free with one axis pointing upright with a single reading (to avoid delays). Figure 3 shows the camera's setup. We are provided a Homogeneous Transform Matrix of a cube with respect to the camera (H_{cam}^{cube}). We can

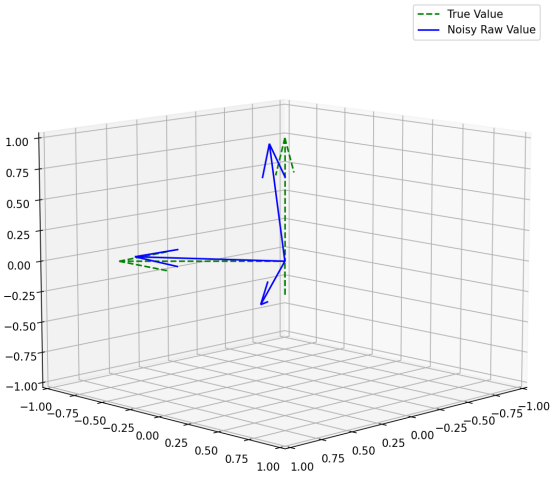


Figure 4: SO(3) Noisy Data and ground truth

calculate the Transformation of cube w.r.t base using equation 3.

$$H_0^{cube} = H_0^{ee} \cdot H_{ee}^{cam} \cdot H_{cam}^{cube} \tag{3}$$

Upon inspection of the simulation code, we found that the data provided was ground truth (Equation 3 was manipulated such that we get H_{cam}^{cube}). Hence, A white Gaussian noise of magnitude $\pm 0.2\ rad$ was deliberately added to the camera pose estimate. The results are shown in figure 4. A simple fix would be a low pass filter with a high cutoff frequency. However, you cannot add 2 noisy rotation matrices and get an SO(3) matrix in the output. We need an approximate SO(3) matrix for our IK to get a solution quickly. Moreover, this estimation has to be quick, and thus we cannot take multiple readings to get reliable data. Hence, a single-shot method was implemented, providing the most robust data possible using some information from the environment.

The only guaranteed information is that one axis will point in $\pm\hat{z}$. Therefore, we modify the R matrix we

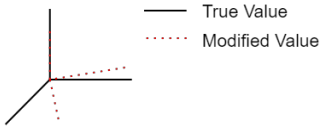


Figure 5: Step 1 of Optimization Problem

obtained from equation 3 such that the closest vertical axis is precisely the same as $\pm\hat{z}$. Then given the other raw values of the axis, we find the closest SO(3) matrix by solving the following optimization problem.

$$\begin{aligned} \text{Given} \quad & R = \begin{bmatrix} r1 & r2 & r3 \end{bmatrix} \\ \text{If } r_2 \text{ is vertical} \quad & R' = \begin{bmatrix} r1 & \pm\hat{z} & r3 \end{bmatrix} \\ \text{Solve for new} \quad & R'' \quad \arg \min_{R''->ortho} ||R'' - R'||^2 \end{aligned} \tag{4}$$

One can solve this problem by simply setting the singular value of R' as identity and making it's determinant 1. This will provide a closest valid SO(3). With regard to programming, we solved the problem by calculating the singular value decomposition and then manually setting the singular values as 1. The newly constructed matrix from this formulation is the R" which is SO(3). The error norm was decreased by a magnitude of approximately 10 from single reading (Figure 6). Coupling this with a low pass filter with multiple readings provides a robust estimate.

2.2 Path Planning

The state estimation module returned the pose estimate of all the blocks while also updating the current state variables of the robot. We then move on to the path planning module. Choosing an appropriate path-planning algorithm helps to ensure safe and effective point-to-point navigation. The optimal algorithm depends primarily on purpose intended. We used this competition as an opportunity to study different planning methods and compare them for pick and place. In this section, we provide a brief analysis of how the following planning methods were performed -

1. Artificial Potential Field planning
2. A* (Search-based planning)
3. RRT (Sampling-based planning)
4. Interpolating curve planning

APF, A*, and RRT have been discussed in class and covered by abundant literature. [11] provides an exhaustive survey of variations of A*, RRT, and other relevant and novel path-planning algorithms.
While these algorithms work incredibly well under complex, constrained situations, the dynamics of the competition made us ponder over many elementary and local planning strategies.

2.2.1 ICP

Where the previous planners work on a global scale, interpolating curve planners use local planning. These methods generate a local path adhering to a curve while considering constraints. Joint configurations for the waypoints thereby generated are found through inverse kinematics and are followed using controllers. We used 3 different types of curves - 3D Lines, quadratic curves and splines using Bezier curves (7).
The bezier curve is a parametric curve defined by a set of control points. As opposed to polygonal lines, bezier curves are smooth and are highly unlikely to cause a 'jerk' during motion

We have generated waypoints keeping in mind the dynamic size of the obstacle, which is updated after every cycle of pick and stack. After many iterations, we decided that combining these curves rather than individual ones would give the best possible results.

2.3 Motion Control

Path planner gives us an optimized, collision-free path to follow, but the controller has to be susceptible to noise, changes in the state of the robot or the target, and inertial momentum, among other problems. Motion planning and path planning fundamentally differ in this regard. We tried and tested 3 different control methods -

1. Position control
2. Velocity control
3. Force Control (APF planning)

Compared with position control, velocity control allows for better performance than position control during a target-reaching task in the 3-D space. A more successful and accurate motion was displayed when the velocity control mode was used, but precision in reaching the target was achieved by position control. Position control also produced more unstable behaviors.

2.3.1 Position IK

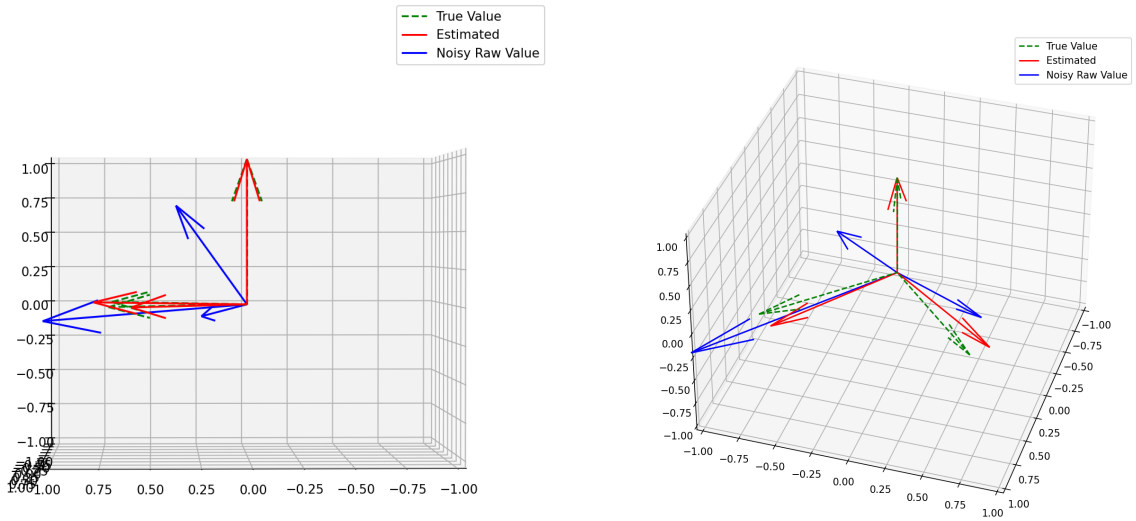
We solved the IK problem using gradient descent. [10] had a good reference for the theory aspect of this module. Gradient descent demanded a good initial guess and often encountered local minima. We hence made sure we had

- Adaptive step size - Step size is larger when the error is large and smaller when closing
- Robust local minima detection - oscillations around a point have been considered. when the error has an oscillatory nature and decreases with time around a fixed point, we tagged it as local minima
- Well-directed random guess - FK is used first to check the feasibility of the guess configuration and if it's well under the constraints of the robot

The steps above helped us considerably reduce the time taken by gradient descent and avoid getting stuck in local minima quite often. moveToPosition() function is used in position control.

2.3.2 Velocity IK

Gradient Descent based inverse kinematics is slow if the seed is not given optimally. Our objective was to reduce the descent time without calculating the seed manually. Moreover, the positional safe commands are



(a) View 1 (b) View 2

Figure 6: Results from single frame

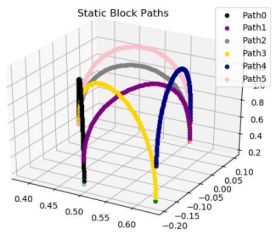
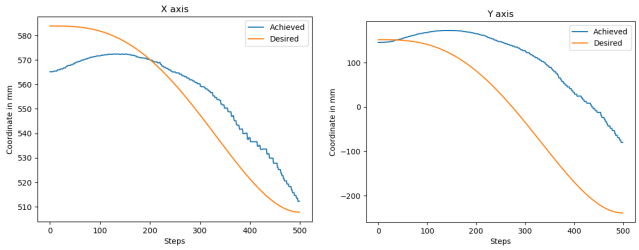


Figure 7: Bezier Curves



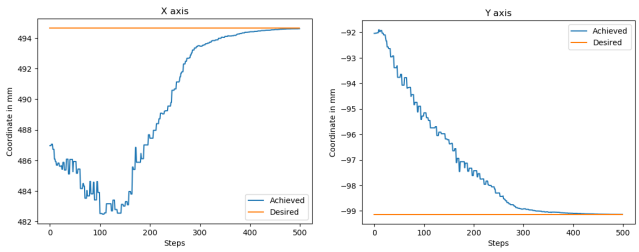
(a) X axis Reference Tracking (b) Y axis Reference Tracking

Figure 8: Reference Tracking

slower than position and angular velocities. However, the angular velocity-based control is not as accurate as plain position control. To address this trade-off, a hybrid strategy was chosen such that initially, the robot uses velocity inverse kinematics, and then as the arm is just above the block, it switches to a position-only controller. The control architecture of the velocity inverse kinematics controller is shown in figure 10. Here we generate our desired position (x) and apply a P controller. We also calculate lateral velocities ($\dot{x}$) and add it in feed-forward which yields $\dot{q}$ using velocity inverse kinematics. We integrate the angular velocity to get the desired angular position which is then passed to the inbuilt controller.

Regarding the pose of the robot, we passed nan values. The final inverse kinematics controlled orientation. We can obtain a better orientation of the robot using IK directly rather than this because we observed that orientation requires a lot more precision than position.

Figure 19 shows the step response of the control scheme described in figure 10. Figure 8 shows the reference tracking of the same. We found that the robot almost converges to the target within 300 steps and hence decided to keep the number of points of trajectory as 300.



(a) Step Response X axis (b) Step Response Y axis

Figure 9: Step Response

2.3.3 Potential Fields

Potential Fields Planner has been covered extensively in previous labs and in class. We basically calculate the attractive force from the block to be picked and stacking position and repulsive force from obstacles which are the entire table plane and stacked tower. We then calculate joint torques using Jacobian, and then we can find the path using gradient descent. How we encounter, and design random walks and local minima have already been described in the course labs.

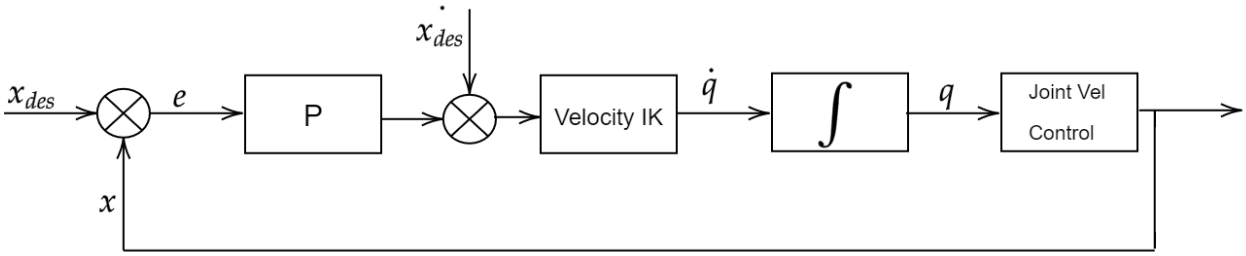


Figure 10: Velocity Inverse Kinematics based Control System

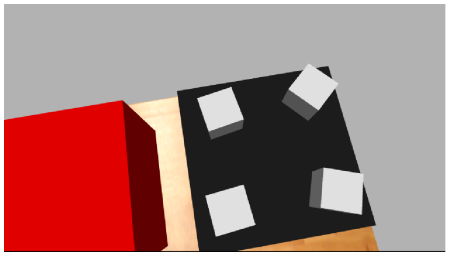


Figure 11: Scout Position

2.4 Static & Dynamic pipeline

The pipeline module interacts with all the other modules to complete the technical aspects of the pick and place task. The pipeline consists of 3 subtasks - Block search, static & dynamic picking, and stacking.

For static blocks, Our Static Pipeline relied on finding all 4 blocks, arranging the order of picking, and performing pick and place. Meanwhile, for dynamic, we prioritized the block in the view that is closest to the gripper to pick and stack

2.4.1 Block Search Algorithm

A good scout position was of utmost importance to ensure all blocks were detected and the rest of the code went untroubled. One example is shown in Fig. 11. We didn't rely on a fixed scout position since the competition could throw improbable situations. Hence, though we had a fixed initial position if, for some reason, the robot didn't detect all 4 blocks, we implemented a block search algorithm where we moved the first and sixth joint such that the field of view changed. We do this 20 times, after which we proceed with what we have in view.

2.4.2 Static Picking

We had numerous options to approach the picking problem thanks to the sub-modules of path planning and motion controllers. We experimented with all possible combinations (the results of which have been detailed in the forthcoming sections) and chose the planner which gave the best time and was also consistent in its results.

- Use RRT / A* / ICP/ gradient descent IK planners to get the path from seed position to stacking position and follow it purely using position control/ Velocity control
- Use IK to get the robot configuration for stacking position and potential field planner to plan and follow simultaneously
- Use a combination of these planners such that we optimally use planners in circumstances that suit them and minimize time and uncertainty

2.4.3 Dynamic Picking

Dynamic picking differs from static picking because the target block moves by a considerable distance (change in position and orientation) between the time we plan a path for it and the time end effector following the path and reaching the picking position. To tackle this situation, we calculate the radius of the circular path where the target block is moving and, thereby, the angular velocity, ω of the block. We then adopted 2 approaches -

- Plan the path for the block in target but while executing the controller, correct the actual next target way point pose matrix by multiplying with the change matrix M, where

$$M = \begin{bmatrix} cos(\omega t) & -sin(\omega t) & 0 & r \cdot sin(\omega t) \\ sin(wt) & cos(\omega t) & 0 & r \cdot (1 - cos(\omega t)) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

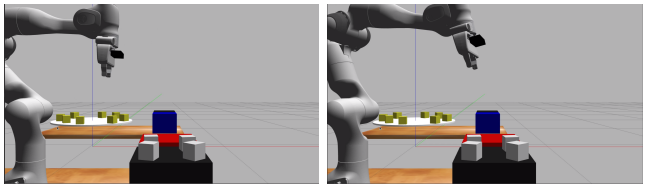
t is time in real-time at each discrete time step for the waypoint following

- Consider an offset time of t1 = Δt. Calculate the M matrix with t1 and ω. Plan for the revised target pose matrix, M multiplied by the original pose matrix. After reaching the revised position and orientation, we take note of the time elapsed t2 and close the gripper after t1 - t2 seconds.

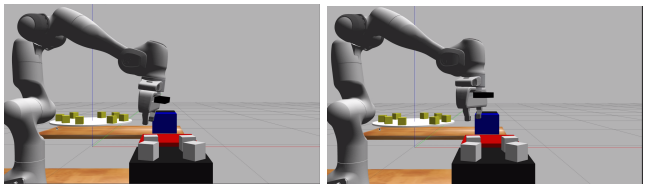
The second approach worked consistently well, given we followed the fail-safe. One major problem we encountered was the gripper losing grip on the block and retreating just above the block. Hence, on the first 'miss' of grip, we plunge down with the jaws open and attempt to grab the block again.

2.4.4 Stacking

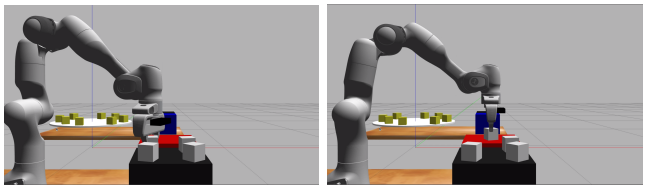
Stacking for static blocks was pretty straightforward. But to cut down on stacking time, we planned optimal trajectories. For the leftmost blocks, we had flat trajectories without taking the block up too high for safety (a mistake that was quite commonly found to be made by other teams). Restricting the safety height to as much as necessary saved time.
With dynamic blocks, while proceeding to stack with the picked-up block, we first move to the scout position and check the gripper jaw position. After we reconfirm that the block has been picked, we plan the path to $0.025 + 0.05 \cdot (n - 1)$ where n is the number of blocks stacked until now.
In both static and dynamic block stacking, paths were planned so that they are strictly approached from the top, and the arm avoids collision with the tower, thereby eliminating the risk of toppling the stacked tower.



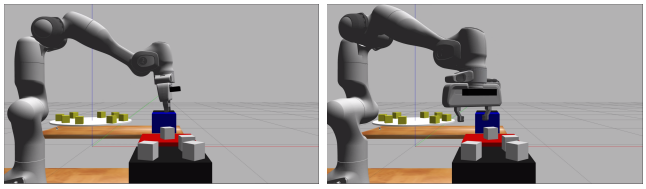
(a) Neutral Position (b) Scout Position



(c) Velocity IK (d) IK for final orientation

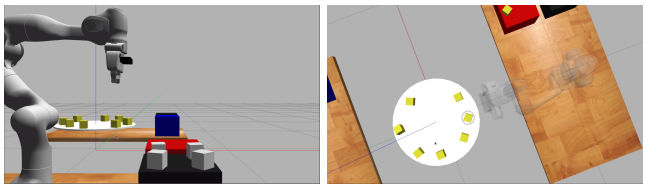


(e) IK on block (f) IK Placing

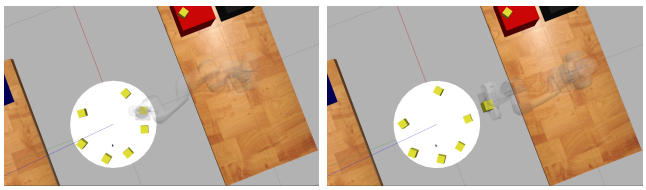


(g) Velocity IK (h) Repeat from (c)

Figure 12: Strategy for Static Blocks



(a) Neutral Position (b) Scout Position



(c) Predicted Intercept Point (d) Pickup

Figure 13: Strategy for Dynamic Blocks

3 Competition Setup & Strategy

3.1 Setup

The task was to maximize the score the robot could reliably achieve during a round of the game. The scorable objects were blocks that had April tags attached to them. Each individual scoreable object supported by a team's goal platform will receive points according to value times altitude, where the altitude is the distance from the center of the object to the surface of the goal platform in millimeters, and the value is 10 for Static Blocks and 20 for Dynamic Blocks. The qualification round lasted 3 minutes per team, and knock-out matches had 5 minutes.

3.2 Strategy

Based on the methods described above, the fastest method was Bezier curved-based velocity IK coupled with gradient descent inverse kinematics. Figure 12 describes the strategy in steps. First, we generate a path based on the Bezier curve; then we do velocity IK (Fig 12 (c)) - we keep some offset on Z, so we reach just above the block. Then we fix the end effector orientation (fig 12 (d)), followed by going to pick the block (fig 12 (e)). Once picked, we solve the IK for the target position and place the block (fig 12 (f)). Then we again generate a Bezier curve from the current position to the block (fig 12 (g & h)).

This approach was the most optimal one. However, the robot's speed was too fast to be allowed in the competition (deemed "unsafe"). Due to the constraint on time, we had to switch our strategy the day before the competition. We replaced the Jacobian-based Control with simple inverse kinematics. In total, we now had to solve IK 6 times for a single block compared to only 2 times in the previous approach. The approach for dynamic blocks is the same as the second approach described in 2.4.3.

We carried out a software benchmark and found that instead of opening the gripper all the way if you

open the gripper to 0.07 meters (diagonal length of the block), you can save 20 seconds of your time. Moreover, in safe to move position, there was a parameter of the threshold for tolerance; we increased the threshold for moving across the space, and while picking, we kept the default threshold. This also boosted our time by 10 seconds.

Regarding the blocks, we decided that in qualifiers, we would pick 4 static blocks and 1 dynamic block on top. However, depending on the performance, if we had 5 minutes of time, we decided to go for 3 statics + 2 dynamics and then 1 static to maximize our score.

3.3 Failsafe

Due to limited hardware testing by our group, we had to rely on the software to handle any anomaly. Hence we incorporated a lot of sanity checks and error handling in our functions.

1. Static Block Transformation Matrix
   In order to ensure that we have 4 static blocks, we implemented a block search algorithm. Moreover, if the end effector is moving with fast velocity and the function is called, it would warn about the possibility of blurry images. In addition, we modified the transformation matrix such that one of the axes is always in $-\hat{z}$. Although our algorithm described in 2.1.2 handles that, there were instances where it failed. Hence, we had a sanity check whether the transformation matrix is a right-hand coordinate system and the axis is pointing downwards.

   If the check fails, the robot searches for a better scout position over the blocks and recalculates the static block matrices.

2. Redundant Dynamic Block Calculations
   We had two different variables for storing the dynamic blocks. One variable stored them all in memory and then deletes the block if it goes out of view for 0.5 seconds. The other variables store the instantaneous blocks and its tag in a dictionary. The reason for redundancy is that if there were any issues in one of them, we switched it to the other variable.

3. Timeouts on the Static Block
   We also had a failsafe that if the static block pipeline runs for more than 2 minutes, we switch to dynamic block. The reason is that we wanted to stack at least 1 dynamic block in the competition, which took 45 seconds in simulation.

4. Optimizing the last angle
   Inverse Kinematics has been one of this project's most significant pain points. If the solution doesn't converge for any reason, we had to ensure that we can still pick up the block. Figure 12(d) shows the IK where desired orientation is achieved. Since the arm is always pointing downwards, rotating the last angle will change the yaw of the end effector. In case of an incorrect solution, we discretized the last angle from $[0,2 \cdot \pi]$ and then calculated FK for each of the iterations1. The one with the minimum error would be a configuration with which the algorithm will proceed.

5. Understanding fails using gripper state
   we checked whether the object had been grasped using the gripper jaw positions. While grasping, we proceed with the stacking planner only if the jaws do not close completely. we also check the jaw positions after it picks and moves up to be sure it didn't drop in between and to know if the block has been caught by its diagonal (in which case, we adjust our stacking position accordingly)

3.4 Code Architecture

Figure 14 shows the code architecture for the implementation. Each module in the diagram (Planner and Motion Control) has multiple types of implementations listed. One can use any of them, and the code will still work as intended making it much more modular.

4 Evaluation

We evaluated our results extensively for static blocks as we wanted to minimize our time for stacking them. Dynamic Blocks could take as much time as they want, so the priority in dynamic blocks was to just work correctly. Regarding static blocks, we added dataloggers to our code to get the joint profiles and execution time.

4.1 Simulation

For evaluating the methods, we tested the stacking for three static & 3 dynamic blocks all in different orientations with robot starting from a random position. Our primary aim was to benchmark the execution time of the planners and the time to stack. The total time for all the methods on 1 block is provided in table 1.
During our testing, we didn't find the static block pick and place faulty or the need to check for fail safes but dynamic failed quite often. Hence in table 2, case 1 is when block is picked up on first attempt without failing and case 2 is when block fails for the first attempt and either re grasps at the same position/ plunges down/ goes back to scout position and then plunges. The average time taken after numerous test runs have been recorded

Method	**Test 1**	**Test 2**	**Test 3**	**Time(s)**
A*	29.103	31.224	25.78	
Artifical Potential Fields	30.7	35.067	31.96	
RRT	30.05	27.45	30.4	
Bezier and Jacobian based planning	13.67	14.23	13.98	
Simple IK	22.13	24.023	23.91	

Table 1: Time Per Static Block for different Methods

We also measured the success rate of each of these methods, however the success rate in simulation is usually 100% if implemented properly.

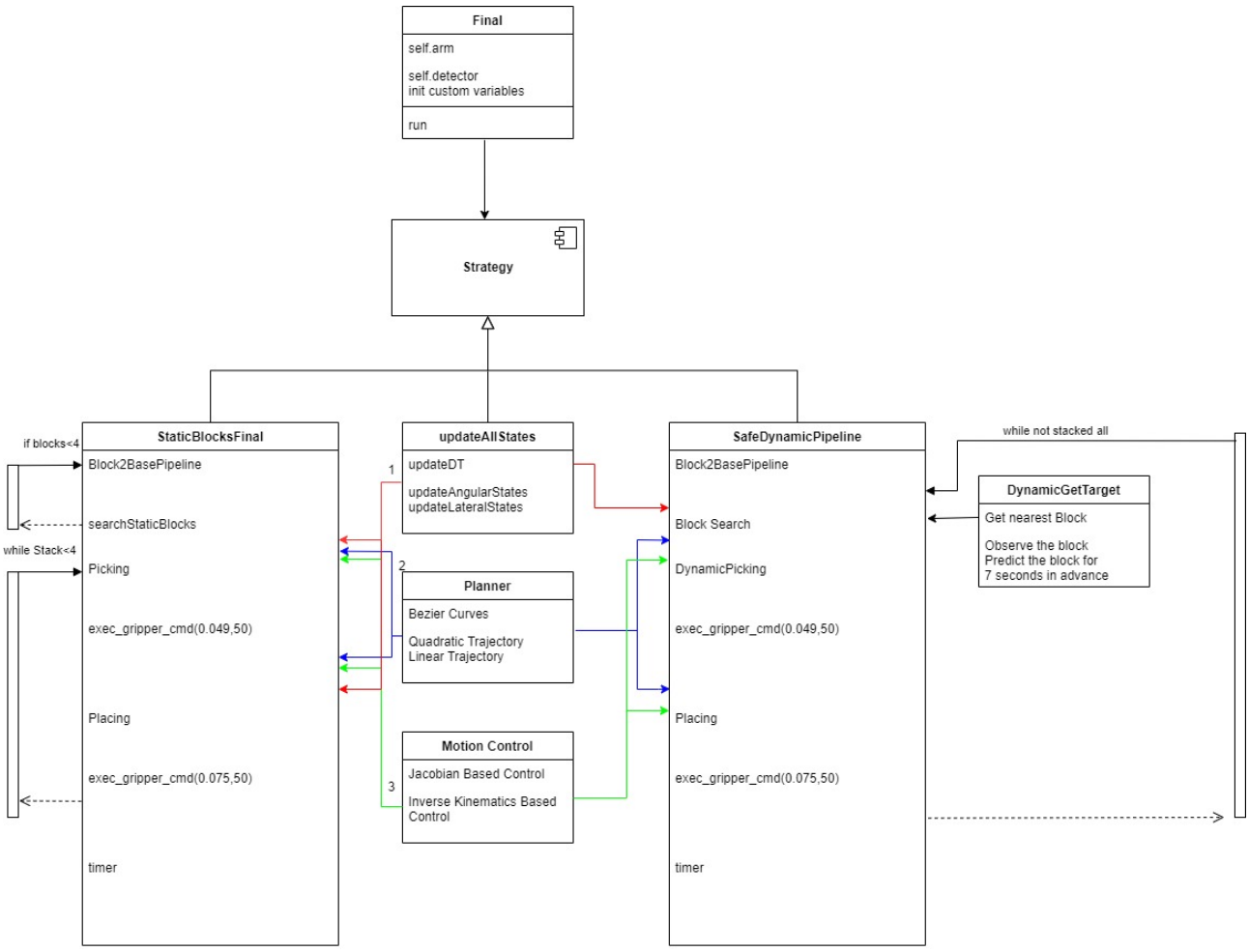


Figure 14: Code Architecture

Method	Case 1	Case 2	Time(s)
A*	36.3	45.4	
Artifical Potential Fields	38.7	44.7	
RRT	38.05	47.8	
Bezier and Jacobian based planning	25.2	35.3	
Simple IK	30.66	38	

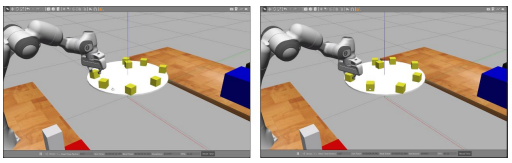
Table 2: Time Per dynamic Block for different Methods

4.2 Hardware

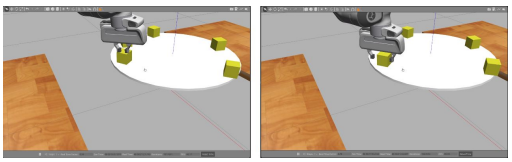
The competition was a perfect stage to evaluate our code's performance on actual hardware setup. We went into the competition with perfect simulation results with hundreds of test runs. Our strategies worked well on both static and dynamic blocks and we clocked good timings. We were equipped with different strategies too, in case of an extreme opponent.

- We started off well in the qualification rounds by stacking the static blocks correctly in approximately 93s. There was considerable difference in stacking time in simulation and hardware. Around 7-8s were wasted in doing block search for static blocks as the robot didn't get a good view of all the blocks.
- Since inertial effects, friction is not well depicted in simulation, real time stacking is more flawed and consumes more time. The misalignment of the stacked tower is an example of minor changes in orientation and effects of inertial movement of the arm
- Dynamic blocks was a real stop block in hardware run. There was a huge difference when it came to predicting the blocks motion and planning for the 'virtual object'. In the qualification rounds, we made a very silly error of keeping the delta t (explained earlier - time taken for arm to plan and reach the block) very low due to which, by the time arm reached the table, block had already left the point of interception.
- The change matrix couldn't also correctly capture the change in orientation and hence the end effector doesn't accurately approach the dynamic block in correct orientation

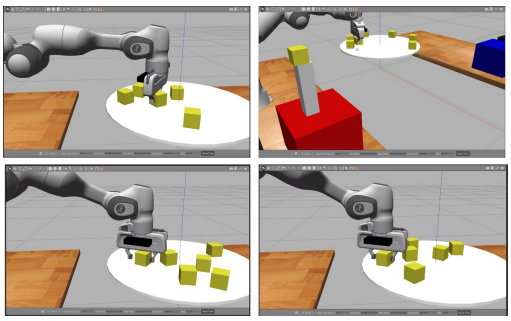
We lost in the quarterfinal of the tournament due to the gradient descent IK giving an infeasible solution and the robot became standstill. We were not able to recreate the error again but a consensus among us is that better Local minima recognition would have made sure such an error wouldn't have occurred



(a)

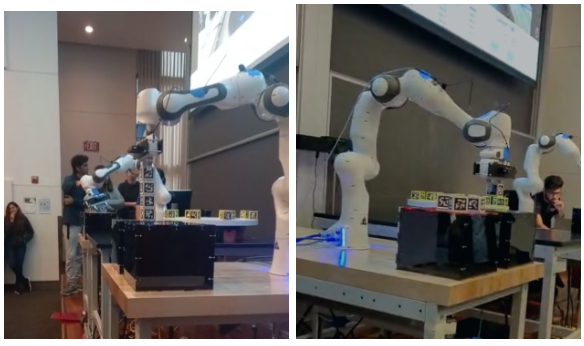


(b)



(c)

Figure 15: Dynamic blocks fail safe performance



(a) On blue team (b) On Red team

Figure 16: Competition performance

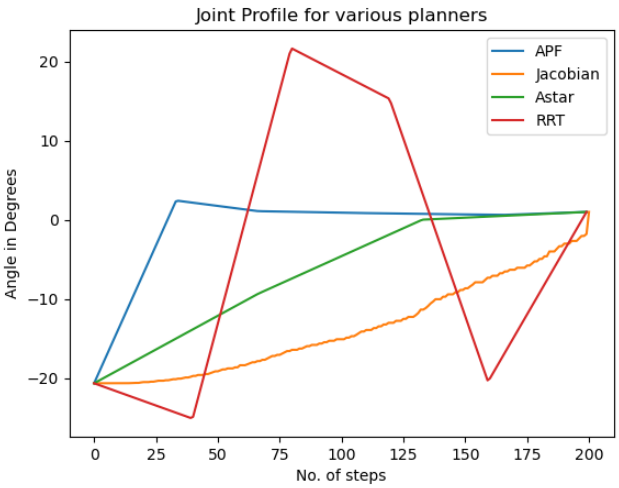
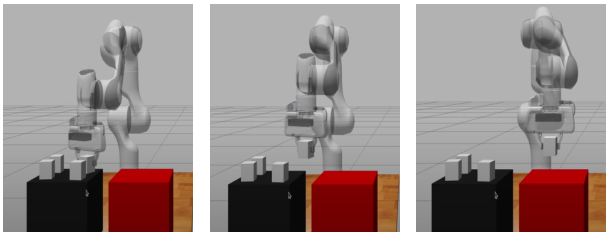


Figure 17: Joint Profiles for various planners



(a) (b) (c)

Figure 18: Static block safety height

natives and fall backs to simple Jacobian/position control

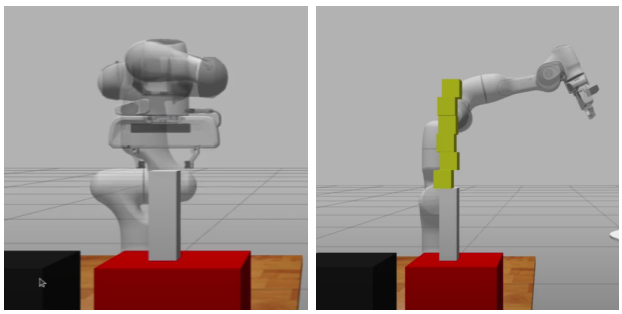
- Jacobian control, on closer inspection will reveal that the curve is not smooth and therefore caused vibrations and vigorous movement of the arm. This results in lots of energy being consumed and hence is not ideal. But the most optimal path was indeed given by curve interpolation planner with jacobian control

5 Analysis & Takeaways

5.1 Planning

We inspected joint profiles for each of these methods to better assess the results. Fig 17 shows the joint profile for second joint.

- RRT clearly has shown that it isn't a good choice for a planner when sampling space is huge and obstacle area is less. Since it is random sampling, while it did give optimal paths in some occasions, it wasn't very consistent and we didn't want to go into the competition relying on it.
- We found, on further investigation after the competition that A* and potential field planners with minor corrections in better heuristics and local minima recognition, could have been great alter-



(a) static (b) dynamic

Figure 19: Final stack performance

5.2 Picking & stacking

As shown in the final stack performance in fig 17, our position and orientation control of the robot was on

target and hence could achieve a perfect stack of static blocks with zero misalignment. Even on hardware (fig 14), the static cube was near perfect with very minor misalignment due to inertial effects.
For dynamic blocks, our fail safes worked quite well as shown in fig 18. Almost 30% of the times, we were successful in picking dynamic blocks only due to the fail safes - plunging down when gripper misses, reattempt for grasp, multiple blocks horizontally/ vertically among many other situations
The primary reason for why fail safes happened quite often was because we were planning for virtual object and approached the block vertically (from above). we hence had very little margin of error to perfectly pick the block. A better alternative would have been to approach it horizontally over the table so that other blocks could be knocked over and we had enough time margin to catch the block even if it fails (since the block wouldn't move)
We unfortunately couldn't perform enough hardware testing to reiterate our simulation results but have analyzed it using various methods to prove its credibility

5.3 Lessons learnt

- A big chunk of our time went in debugging code that is deployed in real time. We learnt about the process - computations done online & offline, how we can cut down on computational complexity and time taken by simple code refactoring
- Perception systems can often be quite tricky to work with. Use of filters and methods such as Procrustes' are crucial since we want to eliminate noise but also make sure that the properties of entity remain same, for ex, orthogonality in case of rotation matrix.
- While we removed noise from the vision input, we introduced noise into the system through planned paths to check the robustness of the code against adverse situations. Simulation results are deceiving and hence we would need better testing strategies going into competitions like these.

Code design is the most important step in a group project. Designing the project's code architecture and untangling the problem into minor tasks is extremely necessary and plays an important role in making sure we test our work enough

6 Conclusion & future scope

The MEAM 520 Pick and place challenge was a fantastic learning opportunity. Through this project, we analyzed the forward, inverse position, and velocity kinematics of a 7 DOF Franka panda arm. We studied and implemented various path-planning algorithms and compared the results to find an optimal algorithm. Controlling the robot to follow a path has also been found to be a challenging task. Though we didn't finish the competition with the best of results, we analysed our performance and redid our work to complete a comprehensive survey of pick and place task by a robotic arm.
The work can be extended in quite many exciting directions -

- A lot of better planning combinations could be come up with to get a more optimal path. Orientation change should be planned taking into consideration the 'offset wrist'. Noise introduced into the system should also be modelled accurately. These factors would play a key role in picking up a dynamic block precisely
- Use ML based IK solver which can be trained on data offline, handle multiple solutions efficiently and hence could be much more reliable, consistent and much faster during real time deployment of the task

References

[1] Sébastien Mick, Daniel Cattaert, Florent Paclet, Pierre-Yves Oudeyer and Aymar de Rugy. Performance and Usability of Various Robotic Arm Control Modes from Human Force Signals

[2] Shangpei Li; Zhijie Wang; Qi Zhang; Fang Han. "Solving Inverse Kinematics Model for 7-DoF Robot Arms Based on Space Vector," 2018 International Conference on Control and Robots (ICCR), 2018, pp. 1-5, doi: 10.1109/ICCR.2018.8534498.

[3] Rutong Dou, Shenbo Yu, Wenyang Li, Peng Chen, Pengpeng Xia, Fengchen Zhai, Hiroshi Yokoi, Yinlai Jiang, Inverse kinematics for a 7-DOF humanoid robotic arm with joint limit and end pose coupling, Mechanism and Machine Theory, Volume 169, 2022, 104637, ISSN 0094-114X

[4] Shi X, Guo Y, Chen X, Chen Z, Yang Z. Kinematics and Singularity Analysis of a 7-DOF Redundant Manipulator. Sensors (Basel). 2021 Oct 31;21(21):7257. doi: 10.3390/s21217257

[5] Klanke, Stefan Lebedev, Dmitry Haschke, Robert Steil, Jochen Ritter, Helge. (2006). Dynamic Path Planning for a 7-DOF Robot Arm. IEEE International Conference on Intelligent Robots and Systems. 3879 - 3884. 10.1109/IROS.2006.281798.

[6] Z. Liu et al., "An Optimal Motion Planning Method of 7-DOF Robotic Arm for Upper Limb Movement Assistance," 2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), 2019, pp. 277-282, doi: 10.1109/AIM.2019.8868594.

[7] Aurelio Piazzi, Antonio Visioli. Global Minimum-Jerk Trajectory Planning of Robot Manipulators. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 47, NO. 1, FEBRUARY 2000

[8] Zheyuan Xie, Zixuan Lan. MEAM 520 Final Project: Vision Based Pick-and-Place

[9] Archit Hardikar, Mingyan Zhou, Jiacheng Song, Haochen Wang : MEAM520 Final Report

[10] Anokhee Chokshi, Yug Ajmera, Divyanshu Sahu, Vanshil Shah : MEAM 520 Final Report

[11] Karur K, Sharma N, Dharmatti C, Siegel JE. A Survey of Path Planning Algorithms for Mobile Robots. Vehicles. 2021; 3(3):448-468

[12] A primer on Bezier curves https://pomax.github.io/bezierinfo/