MEAM 5200 Introduction to Robotics Fall 2025

Final Project Report - Team 5

Wanting Yao 38010041, Chunyu Jiang 49660776, Tianyi Xia 85060087, Yanran Yi 68669276

1 Introduction

In this project, we develop a unified manipulation pipeline capable of autonomously detecting, grasping, and stacking both static and dynamic blocks to maximize the final score under strict safety and interface constraints.

Our approach leverages end-effector–mounted vision to estimate block poses via AprilTag detections and transforms these observations into the robot base frame. For static blocks, the system performs sequential perception-driven pick-and-place actions, aligning end-effector orientation with block geometry and stacking objects incrementally on the goal platform. For dynamic blocks, we implement a predictive grasping strategy that estimates angular velocity from multiple time-stamped observations, forecasts the future block pose, and synchronizes robot motion with the estimated arrival time to enable successful closed-loop interception.

We extensively evaluate our methods in both simulation and the real world. In simulation, our approach achieves a 100% success rate for static blocks and approximately 55% for dynamic blocks. We analyze the failure cases of our methods in both simulation and the real world, and discuss two key lessons learned from this project: the sim-to-real gap and the principle of "less is more."

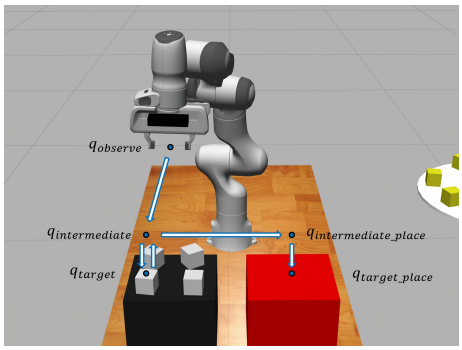
2 Methodology

In this section, we will introduce the algorithmic design and key implementation details for the static block task and the dynamic block task. By integrating perception, kinematics, motion planning, control, and gripper-level feedback, the proposed framework enables robust pick-and-place operations in both stationary and dynamic settings.

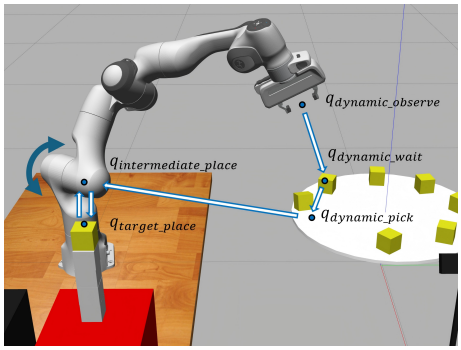
2.1 Static Blocks

The static block pipeline follows a perception-driven, sequential pick-and-place strategy, consisting of system initialization, block detection and filtering, pose transformation and alignment, inverse kinematics-based motion planning, grasp execution with feedback validation, and incremental placement with loop supervision. The objective of the static block task is to reliably detect all stationary blocks located on the static platform, grasp them one by one, and stack them onto the designated goal platform while satisfying safety and workspace constraints.

To achieve this, our algorithm tightly integrates vision-based perception with kinematic planning and low-level gripper feedback. The pipeline begins by moving the robot to a predefined observation pose to ensure robust AprilTag detection. Detected block poses are then transformed into the robot base frame and adjusted to enforce grasp-friendly orientations. For each valid block, pre-grasp and grasp configurations are generated using inverse kinematics to ensure collision-free motion.



(a) Execution loop of the first static block grasp-and-place cycle, starting from $\boldsymbol{q_{observe}}$.



(b) Execution loop of the first dynamic block grasp-and-place cycle, starting from $\boldsymbol{q_{dynamic\_observe}}$.

Figure 1: Overview of static and dynamic pick-and-place execution loops.

Grasping success is subsequently validated using gripper state feedback before the block is placed onto the goal platform with an incrementally updated height. This process repeats until all valid static blocks have been successfully transferred and no further blocks remain.

Fig. 1a illustrates the execution loop of the first grasp-and-place cycle in the static block task, starting from the observation pose $\boldsymbol{q_{observe}}$ and proceeding through grasping and placement.

2.1.1 System Initialization and Observation Setup

For a better view of detection, the robot arm is moved to the static pick observation pose, $\boldsymbol{q_{observe}}$, which is solved by Inverse Kinematics `IK_solver` from the homography transformation matrix we set for the initial observation pose $H_{\text{pick}} = \begin{bmatrix} 1 & 0 & 0 & 0.562 \\ 0 & 1 & 0 & -0.16 \\ 0 & 0 & -1 & 0.60 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and the gripper is opened to the default width as the initialization for the static setting.

2.1.2 Static Block Detection and Filtering

After initialization, the function `choose_all_block()` is called for block detection and selection. The function retrieves all valid detected blocks (supposed to be 4 blocks) from the vision system using `detector.get_detections()`, and filters them to identify valid static blocks for manipulation. Specifically, it excludes blocks that are outside the robot's reachable workspace or located on the wrong side of the table, or simply wrong detections, based on spatial and height constraints.

Since we observed that the real-world camera has a narrower field of view, we set the observation configuration higher than necessary in simulation to ensure that the robot can detect all static blocks at the beginning of the real-world experiments.

2.1.3 Pose Transformation and Orientation Alignment

In this module, the detected block pose is transformed from the camera frame to the robot base frame based on the following formula

$$H_{block}^{base} = H_{ee}^{base} H_{camera}^{ee} H_{block}^{camera}, \tag{1}$$

where H_{block}^{base} denotes the current end-effector pose obtained via forward kinematics, H_{camera}^{ee} is the calibrated camera-to-end-effector transform, and H_{block}^{camera} represents the detected block pose in the

camera frame. Using the function `align_orientation()` and `choose_xy_most_positive()` with the resulting pose, orientation constraints (mainly yaw, which is the rotation in the x-y plane, taken from blocks) are enforced to ensure a physically feasible and stable grasp. The adjustment follows a three-stage pipeline, shown below

- Enforce the z-axis to keep vertical.

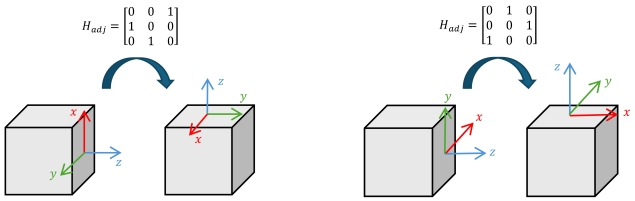


Figure 2: Vertical z-axis constraint during orientation alignment to ensure a stable top-down grasp.

- Normalize three axes and force the z-axis to point down.
- We evaluate four candidate yaw rotations in the horizontal plane and select the orientation that maximizes alignment between the end-effector's planar axis and a team-specific positive reference direction by minimizing the angular deviation from the world-frame axis.

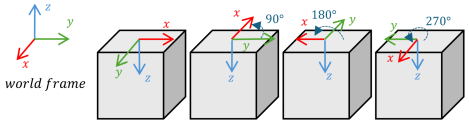


Figure 3: Yaw selection strategy, where four candidate planar rotations are evaluated, and the ideal orientation is selected.

Finally, for implementation in real world, small position offsets are applied to the 4th column of H_{block}^{base} to compensate for perception noise and improve grasp reliability, producing a well-aligned target pose for a higher success rate.

2.1.4 Inverse Kinematics and Motion Planning

In this module, the robot executes the planned grasp motion and determines whether the grasp is successful using gripper feedback. We created the function `grab_a_block()` for this module. The end-effector first moves to a pre-grasp pose $\boldsymbol{q_{intermediate}}$, which is calculated by `IK_solver` from the H_{pick_hi}, to ensure a collision-free approach. The H_{pick_hi} matrix is calculated based on the aligned block pose by adding a vertical offset along the z-axis by $15cm$. Then the robot moves to the target configuration $\boldsymbol{q_{target}}$, which is solved by `IK_solver` from H_{block}^{base} and the gripper is commanded from the preset configuration of position and force. After executing the gripper command, the robot arm will return to $\boldsymbol{q_{intermediate}}$.

To ensure a quick response and feedback from the system, we try two different ways for gripper status checking. The first method is using the provided function `get_gripper_state()` of the `ArmController`, which returns both the position and force of each finger of the gripper. The second method (which is also the original idea of us) is to subscribe to the gripper joint state topic (`/franka_gripper/joint_states`) and monitor the finger joint positions in real time. By recording the gripper opening width after the grasp command is executed, we can determine whether an object

is successfully held between the fingers. If the measured gripper opening exceeds a predefined threshold ($4cm$ in simulation), the grasp is considered successful; otherwise, the grasp is treated as a failure, and recovery behaviors are triggered. The threshold for the real-world setting needs to be set in the lab session and rehearsal session. And the recovery behavior in this module is to re-detect the left blocks and continue grasping the new target.

2.1.5 Incremental Placement and Stacking

In the incremental placement and stacking module, the robot places each successfully grasped static block onto the target platform while maintaining a stable and progressively increasing stack height. We predefined the target position of the first placement, $H_{\text{place\_red}} = \begin{bmatrix} 0 & 1 & 0 & 0.56+\Delta x \\ 1 & 0 & 0 & 0.16+\Delta y \\ 0 & 0 & -1 & z_{\text{target}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ and $H_{\text{place\_blue}} = \begin{bmatrix} 1 & 0 & 0 & 0.56+\Delta x \\ 0 & 1 & 0 & -0.16-\Delta y \\ 0 & 0 & -1 & z_{\text{target}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$ for red and blue settings, respectively. The $\Delta x, \Delta y$ and z_{target} should be modified during the lab and rehearsal session. After a block is successfully placed, the desired placement pose is updated by incrementing the vertical position along the z-axis by $5cm$ to account for previously stacked blocks.

Moreover, to ensure a safer and more stable movement, we align the end-effector orientation with the grasping pose so that the robot does not need to rotate its last joint in the pick-and-place process, which we find is very helpful to save time.

Similarly, we have $\boldsymbol{q_{intermediate\_place}}$ solved from `IK_solver` for a safer placement above the target placement position by $10cm$. After the robot safely moves to the target pose $\boldsymbol{q_{target\_place}}$, the gripper is then opened to release the block with the predefined configuration. Then the robot returns to $q\_intermediate$ and prepares for the next turn.

2.1.6 Loop Control and Termination

In the loop control and termination module, the system supervises the overall static block manipulation process and determines whether additional actions are required. Based on the number of successfully-transferred blocks, if there are more valid blocks available to be picked, which indicates that the number is less than four, the robot will continue another round of execution of all steps introduced above. Otherwise, the static block routine is terminated, and the robot arm will move to the initial pose for dynamic block observation.

2.2 Dynamic Blocks

The dynamic block task extends the static manipulation pipeline to handle objects in continuous motion on a rotating turntable. The objective of this task is to detect, track, and intercept moving blocks by synchronizing robot motion with the predicted future state of the target.

To achieve this, our algorithm integrates vision-based detection, temporal state estimation, motion prediction, and time-aware grasp execution into a closed-loop pipeline. The system first moves the robot to a predefined dynamic observation pose to ensure reliable perception, then selects a suitable dynamic block for interception based on spatial constraints. After transforming the detected block pose into the robot base frame, the block's angular velocity and instantaneous radius are estimated from consecutive observations. Using this information, the future pose of the block at a predefined grasping location is predicted, and a target grasp pose is generated with enforced orientation constraints. Inverse kinematics is subsequently solved to compute a time-synchronized grasp motion, followed by grasp execution, placement, and recovery behaviors. This

perception–prediction–execution loop repeats until termination conditions are met, enabling robust and repeatable dynamic block manipulation under both simulation and real-world settings.

Fig. 1b visualizes a single iteration of the dynamic block manipulation loop.

2.2.1 System Initialization and Dynamic Observation Setup

For dynamic block manipulation, the robot arm is first moved to a predefined dynamic observation pose, $\boldsymbol{q_{dynamic\_observe}}$, which provides a clear and stable view of the rotating turntable. This pose is a manually defined pose in configuration space, separately solved by `IK_solver` for each team side, and ensures that the end-effector-mounted camera continuously observes the motion of dynamic blocks, which should also be slightly modified manually in the configuration space, based on the real-world scene. The gripper is opened to a default configuration, and the system waits until at least one valid dynamic block is detected before proceeding.

2.2.2 Dynamic Block Detection and Selection

Once initialized, the system repeatedly queries the vision module using `detector.get_detections()` to identify dynamic blocks currently visible in the camera frame. Among all detected blocks, a selection strategy `choose_best_dynamic_block()` is applied to choose the most suitable dynamic block for interception. For each detected dynamic block, its position in the robot base frame is evaluated against task-specific spatial constraints to determine whether it lies within a feasible interception region.

In the real-world setting, blocks are filtered based on their vertical position on the turntable, and the block with the smallest forward distance y is selected as the target. In simulation, additional constraints are applied to exclude blocks moving in unfavorable regions of the workspace. Among all candidate blocks that satisfy these constraints, the block with the smallest y is chosen by minimizing a distance metric. This selection strategy ensures consistent tracking of a single dynamic block and improves the robustness of subsequent motion prediction and timed grasp execution.

2.2.3 Pose Transformation and Temporal State Estimation

Similar to the transformation chain in the static pipeline, after a dynamic block is selected, its pose is transformed from the camera frame to the robot base frame using Eq.(1).

To estimate the block's motion, two observations are taken at distinct timestamps (with a 1-second interval). If the target is tracked successfully in two observations, based on the transformed positions of the block at these two time instants, the angular velocity of the block is computed assuming circular motion around the turntable center using Eq.(2),

$$\omega = \frac{\theta_2 - \theta_1}{t_2 - t_1}, \tag{2}$$

where $\theta_1 = \arctan(b_1, a_1), \theta_2 = \arctan(b_2, a_2)$, and t_1, t_2 are two consecutive timestamps of detection attained from `time_in_seconds()`. The block's angular position and velocity are then used to estimate its future arrival time at a predefined grasping location. If the target is missing in the second observation, the system will stop calculating and re-detect for a new target.

Also, we calculated the instantaneous radius of the target block using Eq.(3), and the corresponding variables are shown in Fig. 4.

$$r = \frac{\sqrt{(a_1^2 + b_1^2)} + \sqrt{(a_2^2 + b_2^2)}}{2}, \tag{3}$$

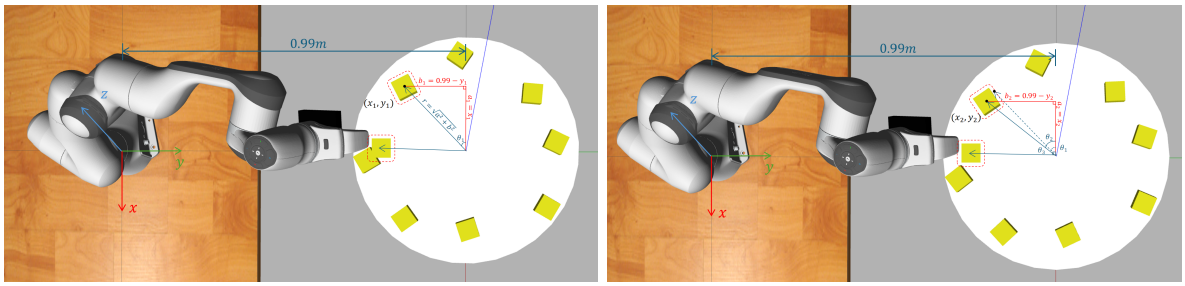


Figure 4: Illustration of the instantaneous radius r estimation for the dynamic target block

2.2.4 Future Pose Prediction and Grasp Pose Generation

Using the estimated angular velocity and the predefined grasping location, $H_{\text{dynamic\_pick\_red}} = \begin{bmatrix} 0 & 1 & 0 & 0.0 \\ 1 & 0 & 0 & 0.75 \\ 0 & 0 & -1 & 0.38 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, and $H_{\text{dynamic\_pick\_blue}} = \begin{bmatrix} 0 & 1 & 0 & 0.0 \\ 1 & 0 & 0 & -0.75 \\ 0 & 0 & -1 & 0.38 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, the future pose of the dynamic block is predicted by propagating its orientation forward in time. Concretely, we first compute the target angular position of the predefined grasping point as $\theta_3 = \arctan(b_3, a_3)$, where $a_3 = H_{dynamic\_pick}[0, 3]$ and $b_3 = 0.99 - H_{dynamic\_pick}[1, 3]$, which need to be measured in real-world scene. Given the current block angular position θ_2 (measured at time t_2) and the estimated angular velocity ω, the expected arrival time of the block at the grasping point is computed as

$$t_{\text{arr}} = \left| \frac{\theta_3 - \theta_2}{\omega} \right|. \tag{4}$$

To reject outliers caused by noisy detections or incorrect velocity estimation, we discard predictions with excessively large arrival time (e.g., $t_{\text{arr}} > 25\,\text{s}$) and re-enter the detection loop.

In addition, we update the grasping point on the fly using the instantaneous turntable radius r from the previous two consecutive observations Eq.(3) to match the observed radius. Specifically, the y-coordinate of the grasping point is updated such that the interception location lies on the same circular trajectory as the moving block, while the grasping height is set to a predefined value z_{dynamic} as Eq.(5)

$$p_{\text{pick}}(r) = \begin{bmatrix} x_{\text{dynamic}} \\ \pm\big(0.99 - r + \Delta y\big) \\ z_{\text{dynamic}} \end{bmatrix}, \tag{5}$$

where $x_{dynamic} = H_{dynamic\_pick}[0, 3]$, and $z_{dynamic} = 0.18m$, which is a preset offset based on the robot's performance. Moreover, the Δy should be measured from the real-world scene, which represents the real difference of distance between the robot base frame to the turntable center.

After determining the predicted time-to-arrival, we propagate the block orientation forward by t_{arr} using a planar rotation about the z-axis and then combine it with the updated grasping translation to obtain the final target grasp pose. Orientation alignment constraints are enforced to keep the end-effector z-axis vertical and ensure a graspable configuration, and a discrete yaw selection is further applied to choose a collision-safe wrist orientation using the function `predict_block_pose_and_align()` following Eq.(6).

$$\begin{aligned} R_{\text{pred}} &= R(t_1)\, R_z(\omega\, t_{\text{arr}}), \\ H_{\text{pred}} &= \begin{bmatrix} R_{\text{pred}} & p_{\text{pick}}(r) \\ 0^\top & 1 \end{bmatrix}, \end{aligned} \tag{6}$$

2.2.5 Inverse Kinematics and Motion Planning

Given the predicted grasp pose, inverse kinematics is solved to obtain a target joint configuration for dynamic grasping. A pre-grasp configuration is generated by lifting the end-effector along the vertical axis z by $15cm$ to create a safe waiting pose $\boldsymbol{q_{dynamic\_wait}}$ above the grasp location. The robot first moves to this pre-grasp configuration and remains there until the estimated arrival time of the dynamic block is reached, ensuring temporal synchronization between the robot motion and the block trajectory.

2.2.6 Dynamic Block Placement and Recovery

After a successful dynamic grasp, the robot transports the block to the goal platform and places it using an incremental stacking strategy similar to that of static blocks. Placement poses are adjusted based on the number of successfully placed blocks to ensure stable stacking. To detect successful grasp and placement, we utilize the same strategy as that in the static setting. We detect the gripper state directly after grasping and before placement. If the gripper is detected closed after grasping, it indicates that the gripper doesn't hold anything after grasping, which is marked as a failed grasp. Or if the gripper is detected open before placement, which means the gripper is in the process of transferring the block, which is marked as a successful placement. In the event of placement failure, recovery behaviors are triggered to maintain system robustness, and the robot will directly return to the dynamic observation pose, $\boldsymbol{q_{dynamic\_observe}}$.

2.2.7 Loop Control and Termination

The dynamic block pipeline operates in a continuous loop until termination conditions are met. If repeated grasp attempts fail or no valid dynamic blocks are detected within a reasonable time window, the system safely exits the dynamic block routine. Otherwise, the pipeline repeats the perception–prediction–execution cycle to attempt additional dynamic block manipulations.

3 Challenge and Evaluation

3.1 Metrics

First, we will use those in Tab. 1 as our metrics, which are success rate, stability rate, and max height. For efficiency, we will observe cycle time and recovery success rate. And we will also observe the positioning and pose of each action to make sure the coordinate system is working well.

3.2 Evaluation

In general, the evaluation metrics are utilized for evaluation both qualitatively and quantitatively. Concretely, for the static blocks, we will make sure the parameters are fine-tuned each time. And the parameters are predefined poses and the offsets for grasping and placement. Also, we will test the robustness of our algorithms as much as possible.

For the dynamic blocks, we still need to fine-tune parameters for initial poses and offsets. Another parameter for dynamic blocks is the waiting durations. Also, to make our algorithm more stable, we need to try different ways to improve the current performance. And it is important to improve the performance of the angular speed prediction.

More details during evaluation in both simulation and real-time scene are demonstrated in the following **Results and Analysis** part.

Table 1: Evaluation Metrics for Manipulation Performance

Category	Metric	Description
Grasping Performance	Success Rate	Percentage of successful grasp attempts for individual stationary or dynamic blocks.
	Failure Rate	Percentage of grasp attempts that fail due to missed grasp, slippage, or perception error.
Stacking Performance	Stability Rate	Percentage of trials in which 4 blocks are successfully stacked without collapse.
	Max Height	Maximum number of blocks stacked vertically on the goal platform during a trial.
Efficiency	Cycle Time	Average time required to complete one full grasp-and-place operation.
	Recovery Success Rate	Percentage of failed grasp attempts from which the system successfully recovers and starts a new attempt.
Accuracy	Pose Error	Position and orientation error between the target end-effector pose and the actual achieved pose at execution time.

3.3 Challenges and Improvement

The challenges we encountered during tests and the corresponding improvements will be introduced.

3.3.1 Static Setting

Challenges We introduce several major challenges in the static setting.

1. First, uncalibrated offsets and non-ideal gripper behavior, or visual detection interference (such as occlusion or detection jitter), can make the robot pick inaccurately.
2. Errors will be accumulated during grasp and placement, which may cause the stacked tower to lean or collapse.
3. The overall speed is not fast enough for a higher score in the competition.
4. Finally, because of these issues, the placement location may need to be adjusted dynamically as we refine our strategy.

Improvements Improvements are presented in the same order as the challenges discussed above.

1. We introduce offset terms into the estimated homography transformation matrix H_{block}^{base} and the position for placement to compensate for intrinsic camera–robot calibration errors and other systematic errors in the system. These offsets must be carefully and accurately calibrated in real-world experimental settings; therefore, to simplify the calibration procedure, the number of offset parameters is deliberately kept limited.

2. To enhance grasp and placement robustness, we explicitly control the gripper orientation throughout the manipulation process. The gripper is first aligned with the detected block orientation during grasping. During placement, the gripper orientation is set to $R_{place} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ to align the end-effector with the positive y-direction. Once the stacking height exceeds approximately $0.56m$, the gripper orientation is switched to an alternative configuration to ensure sufficient clearance and stable stacking.

3. Under safe operating conditions, we reduce the number of intermediate poses from six to four.

4. We design four task execution strategies on top of the dynamic block manipulation pipeline, considering different sequences of static and dynamic tasks and different placement positions. But the only one that was tested in the real world is the first strategy that adopts a straightforward approach, in which all static and dynamic blocks are sequentially grasped and stacked at the center of the target table.

3.3.2 Dynamic Setting

Challenges We introduce several major challenges in the dynamic setting.

1. We often experience unsuccessful continuous block tracking. Missed or false detections can cause the tracker to lose the block or produce unstable measurements, as the current system and algorithm are unstable. This can also lead to angular velocity prediction.

2. Grasp time needs to be more accurate. Since it takes time for the gripper to take action, we need to modify the grasp time based on the real-world performance; otherwise, it will cause collision risk between the gripper and the block.

3. Due to inaccurate prediction, and since our grasp algorithm for dynamic blocks requires high precision of command, the gripper is easy to collide with the block with inaccurate orientation.

4. The overall speed is not fast enough for a higher score in the competition.

5. The gripper status is different between the simulation and different real scenes, which will lead to wrong detection of success.

Improvements Improvements are presented in the same order as the challenges discussed above.

1. First, we fixed the bug in the angular velocity estimation pipeline, where erroneous detections could lead to incorrect θ calculations and consequently inaccurate arrival time predictions. Second, we discard target candidates whose estimated arrival time exceeds 25 seconds, as detections of objects far from the camera are more prone to noise and geometric errors. Moreover, we filter out obviously invalid angular velocity ω estimates to prevent unsafe or unstable robot behaviors during execution. Also, it is also a useful way to select a good observation pose for a better field of view.

2. It is necessary to obtain the real distance from the robot base to the center of the turntable, which will help a lot for all the following calculations. Also, an offset for waiting time might be added according to the performance to compensate for any other potential errors.

3. The end-effector's orientation is aligned to the predicted target pose using the function `predict_block_pose_and_align()`, which extrapolates the block orientation to the estimated arrival time t_{arr} under a constant angular velocity assumption.

4. We removed one intermediate pose while modifying others to maintain stable operations.

5. We created a data logging module to record all essential data through the task process, including the gripper status, to ensure we can track all errors and parameters to be modified.

4 Results and Analysis

4.1 Parameters and Offsets

Here in Tab. 2, we demonstrate all parameters and offsets that need to be fine-tuned in lab and rehearsal sessions for accurate operations and better real-scene performance.

Table 2: Parameter Settings for Red and Blue Robots in Real-World Experiments

Parameter	**Red**	**Blue**
z	0.235	0.235
x offset	0.025	-0.020
y offset	0.035	0.020
Target z	0.190	0.195
Target x offset	0.000	0.000
Target y offset	0.000	0.000
Dynamic z	0.235	0.180
Dynamic x offset	0.000	0.000
Dynamic y offset	0.010	0.005
Wait time (s)	2.7	1.0
Lift-up distance	0.15	0.15

4.2 Simulation

In simulation, we conducted exhaustive experiments on both the red and blue robots and report the running time, success rate (SR), scores, as well as failure cases for dynamic blocks over five rounds in Tab. 3. Our static blocks pick-and-place strategy achieves 100% success rate in all 10 runs. While our dynamic blocks grasping strategy achieves success rates of 58.3% and 54.5% on red and blue robots, respectively. This high success rate for static blocks grasping demonstrates the robustness of our static strategy and the feasibility of our prediction-based dynamic strategy.

Table 3: Performance Comparison on Static and Dynamic Block Manipulation

Robot	Rd	Static Blocks				Dynamic Blocks			Score	Avg. Score
		Time(s)	SR	Cycle Time(s)	SR	SR-5 min	Failure Cases	SR		
Red	1	111.07	4/4	27.79	100%	3/5	due to recover: 2	58.3%	20500	19200
	2	111.84	4/4			3/5	fail to grab: 1; recover: 1		20500	
	3	111.35	4/4			3/4	fail to grab: 1		20500	
	4	109.87	4/4			2/5	fail to grab: 2; fail to place: 1		14000	
	5	111.59	4/4			3/5	fail to grab: 1; recover: 1		20500	
Blue	1	121.87	4/4	29.49	100%	3/4	fail to grab: 1	54.5%	20500	16800
	2	115.91	4/4			2/4	fail to grab: 1; fail to place: 1		14000	
	3	118.60	4/4			3/5	fail to grab: 2		20500	
	4	115.78	4/4			3/5	recover: 1; fail to place: 1		20500	
	5	117.70	4/4			1/4	fail to grab: 1; fail to place: 2		8500	

Failure Cases Analysis During our experiments, we observe three major kinds of failure cases: (1) due to our recovery strategy; (2) robot fails to grab the dynamic block; and (3) the block drops from the placing pose. Here are the possible reasons that cause these kinds of failures.

1. **Due to recovery strategy.** The algorithm incorrectly determines that the gripper fails to grasp the block and therefore opens the gripper, dropping a block that was actually grasped successfully. We analyzed the `.csv` files recorded from the simulation and found many errors in the gripper state data obtained from the simulator. For example, when the robot successfully grasped a block, the distance between the two gripper fingers was sometimes surprisingly larger than the threshold of $4cm$. This is abnormal because we assume the block is rigid, so the distance between the gripper fingers should be smaller than the side length of the block, which is $5cm$. This behavior may be caused by imperfections in the Gazebo simulation and noise in the gripper sensor data.

2. **Fail to grab.** This failure is directly caused by incorrect prediction of the picking location or an unsuitable waiting time. The underlying cause of both issues is noise in the perception module. Noise during block detection can lead to incorrect angular velocity estimation, which in turn results in improper waiting times and inaccurate orientation prediction. During testing, we found that the first several attempts at dynamic block grasping had higher success rates than later ones. This may be due to error accumulation in the simulation.

3. **Fail to grasp.** This failure is caused by an imperfect grasping pose, which results in unstable placement on top of other blocks. The primary reason is noise in block detection, causing the robot to fail to properly align the gripper with the block's orientation.

Static Blocks Running Time Analysis In Tab. 3, we observe that the blue robot requires more time than the red robot to grasp static blocks. We intentionally changed the blue robot's grasping orientation to demonstrate the effectiveness of our "aligning the grasping and placing orientation" strategy. For the red robot, both the picking and placing poses have the camera facing the positive y-axis. In contrast, the blue robot uses the same placing pose, but its picking pose has the camera facing the positive x-axis. As a result, the blue robot must rotate its last joint more to reach the same placing pose, which leads to increased execution time.

4.3 Real Scene

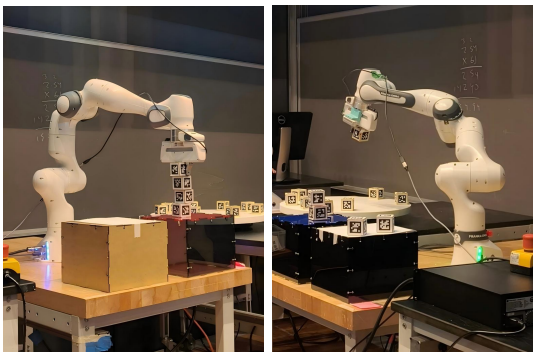


Figure 5: Real robot performance during the competition.

In the real robot competition, we obtained 2250 points and 1500 points on the red and blue robots, respectively, as shown in Fig. 5. Recordings of the final challenge competition are available

at this link. During the competition, the blue robot failed to grasp two static blocks because we encountered issues with the vision pipeline during rehearsal and therefore did not tune its static offset. Despite this, the blue robot successfully grasped two static blocks and two dynamic blocks within the 3-minute time limit and successfully placed one of them. This demonstrates the robustness of our dynamic block grasping strategy in real-world settings.

In contrast, the red robot, which was properly tuned with the static offset, successfully stacked all four static blocks in the challenge. However, it encountered a collision while placing the fourth block. After analyzing the video recordings, we believe this was caused by an incorrect "is placing successful" detection. Specifically, the system may have mistakenly marked the third placement as unsuccessful, causing the robot to attempt to place the fourth block at the same location. This resulted in a collision with the third block, which was unfortunate and prevented us from testing our dynamic grasping strategies.

One reason we achieved significantly higher scores in simulation is that the perception module does not require calibration in the simulated environment. Another reason is that, in simulation, when the robot fails to grasp a block and slightly collides with it, the system can recover from the collision and continue operating.

5 Lessons Learned

One bitter lesson we learned is the sim-to-real gap. Although our robot achieved a 100% success rate grasping and stacking static blocks in simulation, performance degraded significantly on the real robot due to real-world offsets. After setting the offsets properly, we were also able to achieve a nearly 100% success rate in the lab sessions. However, because we were unable to adjust the offsets within the limited rehearsal time, the robot failed to grasp the static blocks of the blue team during the competition. These results emphasize the importance of real-world testing and debugging. In the era of embodied intelligence, bridging the sim-to-real gap is crucial for reliable transfer from simulation to physical robots.

Another essential insight we gained is that "less is more". For the sake of competition, having the robot's gripper directly moved to the rotating turntable and just waiting for the blocks to come is actually a simpler and yet much more efficient approach to grasp the dynamic blocks. Our initial intention was to grasp the dynamic blocks in a more graceful way, including moving to the observation position above the turntable, observing the blocks, calculating the time of arrival, and moving to the grasping position. This approach is indeed elegant and effective, yet it failed to perform efficiently within limited time during the competition.

During the project, the robot reliably executed the commanded actions. However, the detection system performed poorly in real-world conditions, introducing small offsets in the estimated target positions. As a result, although the robot executed the actions correctly, the actions themselves were based on inaccurate inputs, ultimately leading to grasping failures. Furthermore, careful calibration was required for the robot to reach a position with the correct field of view; however, even with calibration, the vision pipeline still failed occasionally.

In conclusion, although it is unfortunate that our carefully tuned algorithm did not achieve the desired performance in the final competition due to offset-related issues, we are deeply grateful for the opportunity provided by this final project. This project allowed us to apply the concepts and techniques learned throughout the course to both simulation and real-world manipulation tasks using the Franka Panda robot. Through extensive system integration, debugging, and real-robot experimentation, we gained valuable hands-on experience and a deeper understanding of the gap between simulation and real-world deployment. Overall, this project significantly strengthened our determination to work on real-world robot problems.