Robot Arm Report

Gene Lam, Charles Yoo,
Will Schlatterer, Daniel Stekol

December 2022

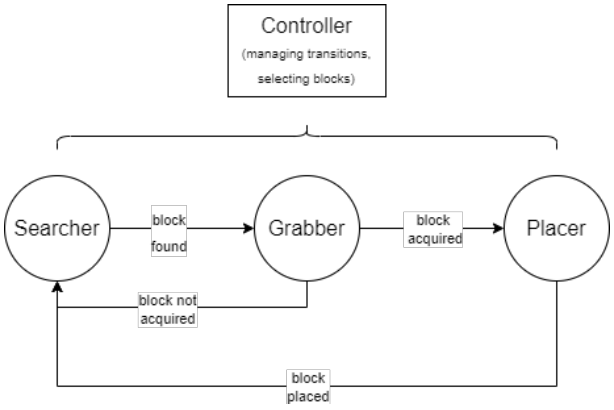
1 Introduction

The purpose of this final project was to design an algorithm for the Panda arm to grab and place blocks on the target platform. There are two sets of blocks: static blocks that remain on a stationary platform and dynamic blocks that move along a spinning turntable. The algorithm is scored based on the number of blocks that it can successfully place on the target platform. An additional reward is given to stacked blocks based on their height within the stack. In this report we will be delving into the methodology behind our code as well as evaluating its performances and analyzing its strengths and limitations.

2 Method

2.1 Overview

To formalize our approach, we model our program as a finite state machine with three states: searching, grabbing, and placing. The robot begins in the searching state, at which point it moves to a suitable vantage point and searches for blocks. It then transitions to a grabbing state, where it attempts to acquire one of the found blocks. Depending on whether or not it successfully grabbed a block, it either returns to the searching state, or transitions to the placing state. Once in the placing state, the robot places the acquired block at the next position in the block stack, and then returns to the searching state.



(1)

In order to implement this approach, we divide our code into four modules: a Searcher, a Grabber, a Placer, and a Controller (which is responsible for managing the three other sub-modules and transitioning between them).

Since the configuration of the environment and the objects it contains are well known, we choose not to use complex and computationally-intensive planning methods such as potential fields, A*, or RRT - instead, we make use of 6-DOF inverse kinematics solutions and context-specific heuristics, as discussed below.

2.2 Controller

The Controller initializes the Searcher, Grabber, and Placer and controls the higher-level logic for moving between Searcher, Grabber, and Placer methods. It has two main loops that it iterates through: the first loop runs until the robot stacks all four static blocks, and the next stacks four dynamic blocks on top of that. Before entering the static block loop, the controller calls the Searcher method search_static(), which returns a map of the static blocks, where each block in the map has its AprilTag name as the key and the block's transformation matrix as the corresponding value. The controller continues to call this method until the map has all four blocks stored in the map, in case the camera doesn't pick up one of the blocks the first time.

2.2.1 Static Mode

After calculating the block transforms for the four static blocks, the Controller enters the Static_Mode loop, which continues to loop while Static_Mode is set to True. Static_Mode remains True while the robot has yet to grab, move, and place (and/or lose) all four static blocks. At the beginning of the loop, the controller selects which block to go for by calling the select_Static_Block(static_blocks) method, which takes in the map of static blocks as a parameter. The method selects a block by iterating through the elements in the map and checking if each block has been placed or lost. The Controller method get_Placed_Blocks() accesses the placer method of the same name and obtains a list of the April Tags of all the placed blocks. Likewise, the Controller's get_Lost_Static_Block(static_blocks) method obtains the list of block locks by checking if each block's z position is below 0.200 m, which would make it unreachable (lost) per the competition guidelines. The controller checks this by seeing if the AprilTag of each block matches those stored in the list of placed blocks and the list of lost blocks. If the AprilTag is not found in either list, the block is immediately selected and the AprilTag name is returned by the select_Static_Block(static_blocks) method.

With the block name selected, the Controller then calls the grab_Static_Block(block_transform) method from the Grabber. The transformation matrix of the selected block is passed into the method, and the method will tell the robot to attempt to grab the selected static block. If the robot is successful, the grab_Static_Block(block_transform) returns True, and if unsuccessful, it returns False. The code and logic for this method are explained in greater depth in section 2.4.1 Static Blocks. If the method returned True, the controller would then move on to the Placer, which contains the place_Block(selected_block,is_Dynamic) method. For static blocks, is_Dynamic equals False, which tells the Placer to start the stack from the base of the scoring platform, placing each newly grabbed block onto the top of the stack. This method is explained in more detail in section 2.5.2 Code Explanation. Had the method grab_Static_Block(block_transform) returned False, the Controller would skip the Placer method and return back to the top of the Static_Mode loop.

When the code was run in simulation and on hardware, we did not have issues losing static blocks. Once all four static blocks were stacked onto the scoring platform, Static_Mode would change to False, and the Controller code would move on to the dynamic blocks loop (Dynamic Mode).

2.2.2 Dynamic Mode

Similar to Static Mode, the Dynamic Mode loop begins with the Searcher by calling its search_dynamic() method and returning a map of the dynamic blocks currently visible to the camera. From there, the Controller checks again whether any of the tags it is reading are from placed or lost blocks. When the block is not placed

or lost, the controller checks if the block is on the hemisphere of the table nearest to the robot (based on the robot's team) and if the block is to the left of the robot as it faces the table. This way, the blocks it tries to grab will be moving closer to the robot as the robot goes to grab them.

After it is determined that a target block is in the desired quadrant on the table, the Controller calls the Grabber's grab_Dynamic_Block(block_transform, t) method, which passes in the target block's transformation matrix as will as the current timestamp when the block was located. The time stamp along with the transformation matrix of the block allows the Grabber to calculate a predicted location of the block by the time the robot grabber reaches the table. Further explanation of how those calculations are made can be found in section 2.4.2 Dynamic Blocks. Like the grab_Static_Block(block_transform) method, the grab_Dynamic_Block(block_transform, t) method returns a Boolean based on whether the robot successfully grabbed the block. If it is unsuccessful, the Controller will try the grab_Dynamic_Block(block_transform, t) method with any other blocks in dynamic_blocks that are within the desired quadrant. If the method returns False for all possible target blocks in that quadrant, then the Controller code returns to the top of the Dynamic Mode loop. If the method is successful and returns True, then the place_Block(selected_block,is_Dynamic) method from the Placer is called again; however, this time, is_Dynamic is passed as True. This method, and how it operates differently for dynamic blocks compared to static blocks, is explained in more detail in section 2.5.2 Code Explanation.

The Dynamic Mode loop repeats until four dynamic blocks are placed on top of the four-stack of static blocks, which if successful in the 5-minute time period in the knockout rounds would yield 28,000 points.

2.3 Searcher

The Searcher is in charge of sensing the environment and reporting the block location and orientation. To do this, it makes use of the AprilTags on each of the blocks and the camera mounted onto the end effector of the Panda arm.

The Panda arm is first instructed to move to a designated configuration depending on whether the team is red or blue and whether the controller is in static or dynamic mode. The Searcher then takes the block pose, which contains location and orientation information, and converts it from camera frame to Panda base frame. In doing so, we are able to get the block pose as it relates to the arm for grabbing and placing. The equation for the conversion is shown below. H^{ee}_{camera} is the transformation matrix converting from camera frame to Panda end effector frame provided by the detector, and T^0_{ee} is the transformation matrix converting from end effector frame to the base frame provided by our Forward Kinematics solution in Lab 1.

$$pose^0 = T^0_{ee} H^{ee}_{camera} pose^{camera} \tag{2}$$

In addition to getting our block pose in Panda base frame, we also align the Z direction of our blocks so that they are facing up. This is because the Grabber will be conducting the arm to approach the blocks top-down for both static and dynamic blocks. To do this, the Searcher selects the column from the rotation matrix that most closely aligns with $< 0, 0, 1 >$. One of the other two columns are selected as our x vector; it does not matter which column is the x vector as the robot can approach the block from either axis. We then set the third column of the rotation matrix to $< 0, 0, 1 >$, and the y vector is then the cross product of $< 0, 0, 1 >$ and the x vector.

In dynamic mode, the Searcher also returns the time stamp in which the detector perceived the block pose. This is used by the Grabber to predict where the blocks will end up on the rotating table.

2.4 Grabber

The Grabber follows a simple waypoint-based planning strategy, making heavy use of the Inverse Kinematics solutions provided by Lab 2.

2.4.1 Static Blocks

The Grabber is passed a 4×4 transformation matrix T (in the frame of the robot base) corresponding to the pose of the target block. In order to avoid collisions with the table even in the face of sensor noise, the grabber sets the target z position to 0.225, since all block origins are known to be at this height:
$T_{3,4} := 0.225$
First, the Grabber ensures the end effector gripper is open, and then moves to a hover position T_h directly above the target block, at a height of 0.3m. The corresponding joint configuration is found by taking the given block pose matrix, overwriting the element corresponding to the z position with 0.3 to obtain an intermediate matrix T'_h; then, since there are four possible orientations of the end effector which would allow it to grasp the block by the sides (corresponding to the four possible 90 degree rotations around the z axis), we use the Lab 2 IK code to compute solutions for all four of the poses $\{T'_h, R_{90}T'_h, R^2_{90}T'_h, R^3_{90}T'_h\}$ (where $R_{90} = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$) , and selecting the joint configuration whose joint angles are closest to the current joint angles (as measured by the L2 norm). If no IK solutions are found, the Grabber immediately returns that it has failed to grab the target block.
Since the searcher always positions the robot arm at a significant height directly above the static blocks, we can move to the hover position directly without risking collisions. Once the hover position is reached, the Grabber descends directly to the grabbing position T_g. Once again, we use the Lab 2 code to find the corresponding joint angles, selecting whichever of the configurations is closest to the current one. Reaching the pose specified by T_g results in the end effector's fingers being positioned on either side of the block. Since the hover position and grabbing position differ only in height and the IK solution is chosen to minimize changes in joint angles, the transition between the corresponding configurations is a smooth downward motion that maintains end effector orientation and avoids collisions.
Once the robot is in the grabbing position, the end effector gripper is closed. The gripper state is retrieved, and if the gripper fingers are at least 0.03m apart, the Grabber returns that it has successfully grabbed the block - otherwise, it returns that it has failed.

2.4.2 Dynamic Blocks

The approach to grabbing dynamic blocks is essentially the same as with static blocks, with the main difference being that the target block position is not given but rather must be predicted. For dynamic blocks, the Grabber is passed a block pose matrix T (in the frame of the robot base) along with a timestamp t corresponding to when the measurement was taken.
In order to find the predicted grabbing pose T_g, we have manually measured the angular speed of the rotating platform to be $\omega = 0.01$ rotations per second (or $\frac{2\pi}{100}$ radians per second) in hardware and $\frac{1}{155}$ rotations per second in simulation. At compute time, we choose a "forecast time" t_f of 10 seconds from when the measurement was taken, and compute the angle α which the block will have travelled during this time as:
$\alpha = 2\pi\omega t_f$
We then find T_g (in robot frame coordinates) by transforming T to world frame coordinates (that is, shifting the origin along the y axis in either the positive or negative direction depending on the team), applying a counterclockwise rotation around the z axis corresponding to α, and then transforming back to robot base coordinates. That is:
$T_g = (M_r^w)^{-1} R_\alpha M_r^w T$

where $M_r^w = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.99 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ for the red team and $M_r^w = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -0.99 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ for the blue team.

Before going to the grab position, we move to a hover position T_h, which as before is simply T_g but with a z position of 0.3. As before, we obtain all IK solutions for the four possible 90 degree rotations of T_h about the z axis, but rather than selecting from the purely based on the joint angle distance from the current configuration, we first select all IK solutions whose angles for joints 1 through 6 minimize the joint angle distances from the current configuration (there are generally several of these since joint 7 can rotate independently to achieve multiple feasible orientations), and then select from these based on which angle for joint 7 results in the lowest cosine similarity between the x axis of the end effector and the radial vector of the turntable:

$q_7 = \arg\min_{q_7'} \frac{\vec{x_e} \cdot \vec{r}}{|\vec{x_e}||\vec{r}|}$

where $\vec{x_e}$ is the unit vector representing the end effector's x axis, and $\vec{r}$ is radial vector (in other words, the normalized displacement vector along the world xy plane between the end effector and the table center). The result of this selection criterion is that the end effector's fingers open and close as much as possible along the radius of the turntable rather than tangentially to it. This means a slight error in timing has less chances of causing a collision, since the block will simply either pass between the open fingers if they close too late, or bump gently into the closed fingers if they close early. In contrast, if the gripper fingers were to open and close tangentially to the circle (approximately along the trajectory of the block), beginning the descent early or late will cause a top-down collision between the block and one of the gripper fingers.

Once the robot has reached the hover configuration, it waits until $t_h = 5$ seconds before the forecasted time at which the block should pass beneath it, then begins to descend and close its gripper in the same manner as with static blocks. This "headstart" time is needed to allow the arm to get into position so that it can intercept the block as accurately as possible. Finally, as with the static blocks, the grabber code then returns whether it succeeded or failed based on the distance between its fingers.

Both of the timing parameters t_f and t_h were obtained via trial and error first in simulation and then on hardware.

2.4.3 Code Explanation

The Grabber code is located in the grabber.py file.

The pick_hover_cfg() helper method selects an arm configuration which hovers directly above the target block by minimizing the joint angle rotation (using vert_offset_cfg() and get_closest_ik() as subroutines) , whereas the pick_hover_cfg_dynamic() method does the same but also accounts for the orientation of the gripper fingers relative to the radius of the turntable, using get_direction_score() as a subroutine to compute cosine similarity between the end effector x axis and turntable radial vector.

The grab_static_block() method accepts a "tfm" parameter representing the pose of the target block, uses the pick_hover_cfg() method to move to a hover configuration above the block, descends and grabs the block. It then verifies that the block has been grabbed by calling the is_block_grabbed() helper method (which checks the distance between the gripper fingers) and returns True or False depending on the success status.

The grab_static_block() method also accepts a "tfm" parameter representing the pose of the target block, as well as a "t" parameter representing the time the pose was captured. It then uses predict_pos() to predict the pose of the block, selects and moves to a hover configuration via pick_hover_cfg_dynamic(), waits until just before the block is set to reach the predicted position, and then descends and closes the gripper, returning a boolean indicating success just as with the grab_static_block() method.

2.5 Placer

Once the grabber has successfully picked up a block, the placer is used to place them onto the platform goal. The placement pose of the robot was determined using the forward and inverse kinematic solvers created in labs 1 and 2.

2.5.1 Considerations

When designing the code for the placer, we considered multiple design choices about how to place the blocks. One choice was whether to make one stack of blocks or two. Having one stack would mean that we would maximize the amount of points that we could get. However, this also means we have the risk of having the stack fall over and losing those points. On the other hand, having two stacks would be because this would decrease the height of each stack, making them more stable and less likely to fall over. To make sure that the end-effector would not tip over either pile when placing, it would alternate between placing blocks in each stack. One clear downside to doing it this way was that we would be awarded less points. We did not consider having more than two stacks because this would reduce our score even more and it would be more difficult to make sure that the end-effector would not run into any of the stacks when placing the blocks. After running the code in simulation and on the hardware, we determined that it would be better to place the blocks in a single stack because having a higher score outweighed the risk of the stack falling over.

The next consideration was whether to place the blocks from the top with the end-effector orientation being $\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$ or from the side with the end-effector orientation being $\begin{bmatrix} 0 & 0 & 1 \\ sin\theta & -cos\theta & 0 \\ cos\theta & sin\theta & 0 \end{bmatrix}$, where θ is the counter-clockwise angle about the x-axis of the end-effector frame. When placing the blocks in the center of the platform goal, we noticed that the end-effector could move higher when stacking from the side rather than from the top. However, the inverse kinematic solution for the side approach was more difficult to find using the IK solver than the top approach. One possibility for this is because our IK solver constrains joint 5 to 0, effectively making the robot arm a 6 degree of freedom arm. Had we designed an IK solver to make use of all the joints, finding an inverse kinematic solution for the side approach would have been much easier. Because of this, we decided to go with the top approach.

Another thing we considered was the placement position of the blocks on the platform. Initially, we had decided on placing the blocks on the center of the table at the position (0.562, $\pm$0.169) meters in the x- and y-directions of the robot base frame. The y position of the end-effector with respect to the robot base frame will be positive for red team and negative for the blue team. If any of the blocks happened to fall off of the stack, they would be more likely to stay on the platform if the stack was in the middle. However, when using the top approach for placing the blocks, the arm could only move up to 0.5 meters in the z-direction. This would mean that the arm could only stack up to 6 blocks in the center of the table. To address this problem, we decided to move the stack closer to the robot arm to the position (0.5,$\pm$0.1) meters in the x- and y-directions of the robot base frame. When we did this, the robot arm could now move up to 0.7 meters in the z-direction and could place up to 8 or 9 blocks.

One final consideration was how big the margin of safety should be to make sure that the robot arm does not collide with any blocks while stacking. This was done by making the robot move up in the z-direction from its grabbing position, then moving above the stacking position, and finally moving down in the z-direction to the placing position. For the static blocks, the robot arm is first moved up 0.05(i+1) meters in the z-direction from its grabbing position, where i is the number of blocks that have already been stacked. On the other hand, for dynamic blocks, this strategy would not work because for some grabbing poses, the robot-arm would not be able to move directly upwards in the z-direction. Therefore, for the dynamic blocks, the robot arm is first moved (0, $\pm$0.5, 0.05(i+1)). Then the arm moves above the stack to the position (0.5,

±0.1, 0.23+0.05(i+1)) before moving to the final stacking position (0.5, ±0.1, 0.23+0.05i). We decided to have the placer move like this because it would ensure that the robot arm would not collide with anything while placing the blocks while minimizing excess clearance that would slow down the placing process.

2.5.2 Code Explanation

The placer is defined as a class that can be used in the controller. During initialization, the placer defines attributes for the forward and inverse kinematics solvers. It also defines attributes for the arm controller and team color, which are both passed to the class during initialization. Lastly, it creates an empty list for the blocks that have been placed.

The placer has 2 functions: place_block and get_placed_blocks. place_block is the main function of the placer and is called to place the blocks on the platform. When called, the function takes in the tag for the block it is placing as well as whether or not the block is static or dynamic. It then defines a variable i as the number of blocks that have been stacked, which can be found by looking at the length of the placed block list. If the block is static, the arm's current joint angles are found using the get_positions() function in the arm controller. This is converted to an orientation and a position using the forward kinematics solver. The inverse kinematic solver is used to find the joint angles to move the arm 0.05(i+1) meters above this position. Then the arm moves to this position using the arm controller's safe_move_to_position() function. After this, the inverse kinematics solver is called to find the joint angles to move above the stack at (0.5, ±0.1, 0.23+0.05(i+1)) and to move to place the block on the stack at (0.5, ±0.1, 0.23+0.05i). The y value will be positive if the team color argument is red and negative if the team color is blue. Once this is done, safe_move_to_position() is used to move the arm above the stack, then to place the block and open the gripper using exec_gripper_cmd(0.08,10), and finally back above the stack. Afterwards, it will append the block tag to the list of placed blocks.For dynamic blocks, the process is nearly the same except for the first position it moves to. get_positions() and the forward kinematics solver are used to get the orientation of the end-effector in its grabbing position. The inverse kinematics solver is then used to find the joint angles to move the arm to (0, ±0.5, 0.05(i+1)) while keeping the same end-effector orientation. safe_move_to_position() is then used to move to that position. Then the arm moves above the stack, then to place the block, and finally back above the stack.

get_placed_blocks returns a list of the blocks that have already been placed.

3 Evaluation

3.1 Competition

3.1.1 Strategy

Our strategy going into the competition was to stack our four static blocks within the first two minutes, and then use the remaining minute in the qualifying rounds to hopefully get one dynamic block stacked on top. Despite being able to grab and stack the dynamic blocks successfully in simulation, we only had 30 minutes to test it on hardware and were unsuccessful in getting it working there. We were hoping that the qualifying rounds would provide us with enough additional insight to be able to make better adjustments during the break before the knockout rounds. With those adjustments, our goal for the knockout rounds was to get two dynamic blocks stacked on top of our four static blocks by the end of the extended 5-minute period.

3.1.2 Qualifying Rounds

Our first qualifying round went exactly as we expected it to go. We stacked our four static blocks in 1 min 58 sec, and we spent the last minute unsuccessfully trying to grab a dynamic block. Our orientation and

timing looked good, but our positioning appeared to be too close to the center of the table along the y axis.

In the second qualifying round, our robot was grabbing static blocks and then immediately dropped them. Thankfully, because our print statements showed that the end effector wasn't working (it was returning that the fingers were only 3.7cm apart while holding a block, which was impossible since the blocks are 5cm wide), we were able to notify the TAs of the problem and get the robot rebooted. We also changed the gripper width threshold for determining grab success from 0.04 to 0.03, thus making the code more robust to faulty gripper readings. Once the robot and the round were restarted, the robot was successful in stacking our static blocks again, doing so in 1 min 57 sec. The robot again tried to go for dynamic blocks in the final minute, but its position as it went to grab blocks was off again, this time too far from the table center along the y axis. We hypothesized that the turntable had slightly moved at some point during the competition, so that the blue team robot (from the first round) was slightly too close to the table, and the red team robot (from the second round) was slightly too far away from the table.

Having scored 4,000 points in both qualifying rounds, we won the first seed in our group and moved on to the knockout stage. With the hardware results in mind, we used the break to try make the static block stacking motions more time-efficient, and to fix the offset issues. However, the amount we needed to shift our robot could only be estimated, since those shifts would not work in simulation, and we were not able to test on hardware in time before the knockout rounds commenced.

3.1.3 Knockout Rounds

In the quarterfinal round, we were able to stack our four static blocks in 1 min 55 sec. This was not as big of an improvement as we were hoping for given the adjustments we made during the break, but we were hopeful that the remaining 3 minutes of the knockout rounds would be enough time to stack one or two dynamic blocks on top. However, our round was cut short when our robot's positioning was offset once again, causing the end effector to collide with the top of a dynamic block, and therefore causing our robot to be S-stopped. The opposing team was able to stack two dynamic blocks on top of their four static blocks before the other team's TA mistakenly hit their S-stop while their robot was stacking more blocks, thus ending the round early and eliminating us from the competition.

3.1.4 Final Code Version

Our final code version, which has been updated and resubmitted since the competition, mainly differs from the previous version in the timing parameters (ex. the forecast time for trying to predict the position of dynamic blocks), which have been refined via trial and error to maximize the success rate of the grabber.

3.2 Success Rate

With additional code refinement, we were able to get the dynamic blocks to stack on top of the static blocks. The following table shows our tests and the success rate of arm in grabbing and placing a block on top of a stack. Tests were performed on the blue team in simulation and in lab.

Block Type	Hardware Success Rate ($n_{stat} = 4, n_{dyn} = 4$)	Software Success Rate ($n_{stat} = 4, n_{dyn} = 8$)
Static	100%	100%
Dynamic	100%	87.5%

Due to limitations in lab time, we were only able to get results from the final version of our code once, and it happened to be successful on 100% of the time. We would have liked to gather more data with additional time to test for the robustness and effectiveness of our final solution. The previous version of our

code (with slightly suboptimal timing parameters), which we were able to test on a larger number of blocks, was successful on dynamic blocks 62.5% of the time.

Our static block stacking is successful 100% of the time in both hardware and simulation, with the caveat that the gripper must be returning accurate readings (which was not always the case in competition) - as mentioned, we mitigated this risk by changing the finger distance threshold at which the program concludes that it has successfully grabbed a block.

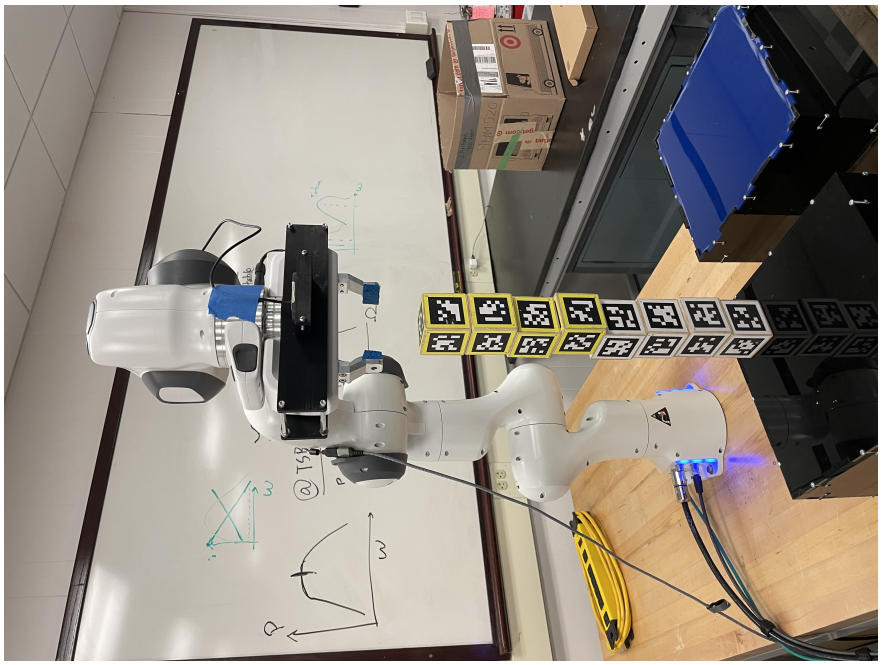


Figure 1: Panda arm stacks 4 dynamic and 4 static blocks before terminating.

3.3 Timing

This section provides the speed performance of our pick-and-place solution on static and dynamic blocks. The observed times at which each block is grabbed and placed since the program start time are shown in the table below. Observed times were measured with tests on the blue team in simulation and in lab. Also note that our program limits the number of stacked blocks to 8 (to minimize the risk that the stack collapses) so although there are multiple dynamic blocks, our solution would place at most 4 dynamic blocks on top of the 4 stacked static blocks.

In the lab, we were able to grab and place all 8 blocks in just over 4 minutes. It took on average 12.25 seconds for the static blocks to be placed after they are grabbed and 13.25 seconds for the dynamic blocks, which can be attributed to the increase in distance for picking and placing dynamic blocks. What's also interesting is the time between when a block was placed and when the next block is grabbed. For static blocks, this time was on average 13 seconds while for dynamic blocks, it came out to 22.25 seconds. Note that the grabbing time is considerably less consistent for dynamic blocks than for static blocks: this is because for dynamic blocks, the grabber must occasionally wait for the turntable to rotate further so that a block enters its reachable workspace.

A limitation with our solution is that we predict where the block will end up, hover over that position using IK6, and then grab the block. This forecast time adds to the waiting period for dynamic blocks. Therefore, if we were to optimize for time, we would experiment with how low we can set the forecast time

	Hardware Wall Time (in seconds)				Software ROS Time (in seconds)			
Block	Grab Time	Δt	Place Time	Δt	Grab Time	Δt	Place Time	Δt
Static 1	14	14	26	12	16.312	16.312	27.359	11.047
Static 2	39	13	52	13	43.246	15.887	54.422	11.176
Static 3	65	13	78	13	70.318	15.896	81.514	11.196
Static 4	91	13	102	11	97.491	15.977	108.707	11.216
Dynamic 1	124	22	136	12	133.132	24.425	147.083	13.951
Dynamic 2	156	20	170	14	170.086	23.003	181.976	11.89
Dynamic 3	194	24	207	13	205.896	23.92	218.762	12.866
Dynamic 4	230	23	244	14	252.162	33.40	264.786	12.624

while still leaving enough time for the arm to get into position.

3.4 Placer Robustness

For the task of placing blocks in the stack, the placer that we designed is sufficient for placing blocks. The placer was very good at placing static blocks and was able to stack them neatly, with little variation in the positioning and orientation of the blocks. However, it had some trouble placing the dynamic blocks. While the positioning was usually correct, the dynamic blocks were not always directly over the previous blocks in the stack. This seems mainly due to variations in how the dynamic block is positioned in the gripper. In rare cases ($< 10\%$), the block would be too far off the stack and would fall off. This posed serious issues because the placer had no way of knowing that the block had fallen off. Thus, when stacking the consecutive blocks, the placer would go too high and drop the blocks from a height above the stack.

One area that the placer is robust is in the movements that it uses when moving to place the block. For static blocks, this is fairly simple as the general positioning of the arm is very similar when picking up the static blocks. In this case, the robot arm easily moved the placed blocks to the goal platform. The more difficult problem was moving the dynamic blocks because there would be more variation in the positioning of the robot arm. To address this, the placer is designed to move to a position that it has been able to consistently reach from any position around the turntable. Additionally, instead of a constant hovering height, the placer uses a varying hovering height that is based on the current height of the stacked blocks. This streamlines the placing process without introducing much risk of knocking the stacked blocks over.

4 Analysis

4.1 Grabbing Strategy

As far as we are aware, we are the only group to successfully attempt retrieving block with a vertical approach rather than a sideways approach. The benefit of this strategy, as shown in the Timing tables above, is that it is quite fast - however, as the competition demonstrated, this approach is also more risky, as small calibration errors are more likely to result in a vertical collision with a block while the end effector is descending. We attempted to mitigate this risk by positioning the gripper fingers parallel to the turntable's radius, but it seems this is not always enough to compensate for real-world errors. Nevertheless, as can be seen in the video linked above, our grabbing strategy is highly effective when it is well-calibrated.

Qualitatively, we observed that the grabber was less accurate with blocks that were very close to the edge of the turntable, likely because the greater linear velocity of blocks which are further from the center of rotation amplifies small timing errors. We attribute inaccuracies in retrieving dynamic blocks in hardware

mainly to measurement noise, whereas inaccuracies in simulation can be attributed to inconsistent simulation run speeds, as well as the dynamic blocks' inexplicable tendency to wobble back and forth on the table while it rotates.

As we also discovered during the competition, the gripper distance measurements are particularly prone to random failures, forcing us to use a very large margin of error when analyzing gripper readings to determine grab success.

4.2 Limitations and Improvements for the Placer Function

The placer is not really robust when it comes to perturbations in the positioning and orientation of the blocks in the gripper. This is not a problem when it comes to static blocks because the positioning and orientation of these blocks is well defined and can be grabbed with little variation. However, for dynamic blocks this becomes a larger issue as there is some uncertainty when predicting where to grab the blocks. Because of this, there is a risk of dynamic blocks being poorly placed and falling off of the stack. If there was a consistent offset in position or orientation of the block within the gripper, then the placer code could be adjusted to take this into account. Since this is not the case, it is very difficult to create a more robust code that can neatly place dynamic blocks without being able to collect data on the orientation and positioning of the block within the grippers. However, one work around to this is placing the dynamic block on the static block starting platform. Once this is done, the searcher and grabber can be used to pick up the block in an optimal position and orientation before placing it on the stack.

Another limitation in the placer is its inability to detect whether or not it has successfully placed a block. When a block would fall off the stack, the placer would continue stacking blocks under the assumption that the previous placement had been successful. As a result, it would release the blocks too high and they would drop from a height above the tower and continue to fall off. In the worst case, this also knocked over the blocks that had been successfully stacked. One way to address this issue would be to use the camera to check that the last block has been properly stacked. This can be done by calculating the position and orientation of the block. If it is within a reasonable amount of variation from the expected values then it has been successfully stacked. If not, then the arm knows where the block is and can restack it properly.

5 Conclusion

Overall, the design of our algorithm has been highly effective at acquiring and stacking both the static and dynamic blocks. While we were not able to successfully grab dynamic blocks during the competition, further tuning was done in lab and we were able to successfully stack 8 blocks. We were also able to significantly cut down on the time that it takes to search for and place blocks. One area that we would like to improve is reducing the waiting time of the grabber code when it is forecasting the position of the dynamic blocks. Another area that we would like to improve would be the robustness of the grabber and placer functions. Currently, the grabber function requires sensitive calibration in order to successfully pick up blocks. The placer has no ways of dealing with off centered blocks and does not know when a block has been successfully placed. If we had more time to continue testing on the hardware, these are the issues that we would most likely target in order to improve the effectiveness of our code.

Unlike previous labs, hardware testing played a much more significant role in the final project. We found that what worked for us in simulation did not exactly translate to success on hardware. This could have been because of noise in camera measurements or variations in the turntable motor speed. Regardless of the causes, testing on hardware gave us much more useful insight into how our algorithm would run during the competition. One useful hardware test would have been running our algorithm on both arms to imitate the competition environment - this would have allowed us to test different strategies (i.e. going for dynamic

first vs static first) against each other to see which ones worked best. Overall, however, we are quite pleased with the speed, efficiency, and accuracy of our final product.