MEAM 520 Final Project Report

Method

The explanation of methodology will begin with an overview of the strategy underlying the proposed solution. When executing the script, 4 goal locations will be defined as a list object at the beginning of the program according to the location of the goal platform. Each goal location is at one corner of the goal platform, 0.06 meters inward horizontally and vertically with respect to each vertex of the corner. After defining the goal locations, the program will loop through the four locations. In each loop, the robot arm will grab one static block and place it at each respective goal location. Then, the nearest dynamic block from the prediction of locations of dynamic blocks will be grabbed and placed on top of the static block. During the process of grabbing and putting the block, the orientation of the block will be changed to make the white face, which is tag 6 for static blocks and tag 12 for dynamic blocks, towards the camera. As a result, 8 blocks will be grabbed and placed on the goal platform in a stacking manner under an ideal scenario theoretically.

The reason for choosing the approach of changing orientation and stacking is it can obtain the highest bonus points according to the final project instructions. The rationale for only trying to pick up and place 8 blocks is from the calculation. After the functionality of picking up and placing static blocks was finished, the program was tested for 5 times, and the time durations for each interaction, which is picking up and placing each block, was recorded:

	Trial 1	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
Block 1	30.51652	30.99315	31.02889	31.90282	32.15491
Block 2	33.67834	33.44743	31.03022	31.84721	29.42452
Block 3	29.74335	32.67815	33.55446	32.19328	33.28724
Block 4	33.96731	31.63762	29.17948	33.53138	33.28729

Table 1 Interaction time with each static block from 5 trials

The average time of interaction is then calculated using the following equation:

$$t_{i\_avg} = \frac{\sum_{T=1}^{5}\sum_{N=1}^{4} B_T^N}{4 \times 5} = 31.95418$$

Therefore, since the final competition will last for 3 minutes each round, the maximum number of blocks that the program will be able to interact with is about 6. Therefore, being able to interact with 8 blocks is already enough. There is no need to interact with more blocks.

To achieve the proposed strategy, several algorithms, including calculations of the homogeneous transformation matrix, inverse kinematics, the transformation of rotation representations, and the transformation of the coordinate system, were implemented in the solution. When transforming points in space between world frame, camera frame, tag frame, and robot base frame, pre-multiplication and post-multiplication are used. When calculating configurations from given points in space, gradient descent is used to find IK solutions. In previous labs, algorithms and concepts of homogeneous transformation matrix and inverse kinematics were already extensively explored. So, this report will not go into these aspects deeply and will mainly talk about implementations of them. The transformation of rotation representations, specifically between rotation matrix and quaternion, is performed when changing the orientation of the

block. A quaternion can be represented as $Q = [q_w, q_x, q_y, q_z] = q_w + q_v$, which is a scalar and a vector respectively, or can be considered as a combination of axis and angle. When offsetting the rotation on the Z-axis, the transformation between the rotation matrix and quaternion is used. Transformation of the coordinate system, specifically between the Cartesian coordinate system and polar coordinate system, is used to predict the location of dynamic blocks.

After an overview of the proposed solution, detailed implementation will be explained. Firstly, to detect tags in the environment, the function *tag_detection()*, which utilizes the *detector.get_detections()* API and returns a dictionary with a combination of tag number and a generated UUID being the key to differentiate each tag considering the possibility of repeated tag number, and the homogeneous transformation matrix of each tag in the camera frame being the value, is defined. The first element of each key is the tag number, and the second element of each key is the generated UUID. Interaction with the static block is the first task when looping through the list of goal locations. To accomplish this task, since the *tag_detection()* function returns homogeneous transformation matrices of each tag in the camera frame, the homogeneous transformation matrices of each detected tag should be converted to robot base frame. In this process, tag0 is acted as the intermediate frame since it is placed in a fixed location relative to the robot arm. Based on this intermediate frame, the homogeneous transformation matrix from the camera frame to the robot base frame can be calculated by homogeneous transformation matrix of tag0 in robot base multiplied by the inverse of the homogeneous matrix of tag0 in the camera frame:

$$H_{Camera}^{Robot\ Base} = H_{Tag0}^{Robot\ Base} H_{Camera}^{Tag0} = H_{Tag0}^{Robot\ Base} (H_{Tag0}^{Camera})^{-1}$$

The calculation is encapsulated into a function called *get_camera_in_robot_base(tag0_in_camera)* with the homogeneous transformation matrix of tag0 in camera frame as the input. Then, *transform_tag_from_cam_frame_to_robot_frame(tags_dict_in_camera, camera_in_robot_base)* is defined. This function loops through the dictionary outputted by *tag_detection()*, calculates the homogeneous transformation matrix of each value in robot arm frame according to the output from *get_camera_in_robot_base(tag0_in_camera)*:

$$H_{Tag}^{Robot\ Base} = H_{Camera}^{Robot\ Base} H_{tag}^{Camera}$$

Then, the joint configurations for reaching homogeneous transformation matrices of each static block and goal locations are calculated using the *inverse(value, seed)* developed in the previous lab, appending to two separated lists. In the first half of each loop, the robot arm firstly moves to the neutral position. The program records the current time and gets the locations of each dynamic block as the arm starts grabbing. Then the arm moves to the configuration of the static block, and execute *exec_gripper_cmd(distance, grab_force)* two times with different pass in the value to grab the block. The program records the current time and gets the locations of each dynamic block again as the arms finish grabbing, and calculates the time takes for grabbing one block. Then, the arm moves to the respective goal location and similarly places the block.

During interaction with the static block, the orientation of the object should be changed to make the white face, which is tag6, upward. Firstly, since each block is randomly placed in the environment, there might be rotation on the Z-axis with arbitrary direction. Therefore, to make the further process of the orientation easier, the rotation on the Z-axis needs to be reset to 0. To do so, q_w and q_z of a quaternion need to be calculated from the rotation matrix to identify the rotated angle of the block:

$$qw = \frac{\sqrt{1 + M_{00} + M_{11} + M_{22}}}{2}$$

$$qz = \frac{M_{10} - M_{01}}{4 * qw}$$

Since the block should be placed on a flat surface, in theory, no rotation can happen on the X-axis and Y-axis, which is also true in practice since the calculated q_x and q_y were always 0 when debugging the program. Therefore, by checking whether q_z is negative or positive, the direction of the offset angle to be applied can be determined. The value of the angle can be calculated using the equation below:

$$\alpha = \cos^{-1}(qw) * 2$$

This equation was derived from the definition of qw:

$$qw = \cos(\frac{\alpha}{2})$$

After resetting the rotation of the block, based on the schematics provided in the final project instructions, rotation on the X-axis and Y-axis to make the white face upward can be easily determined since once the rotation on the Z-axis is reset, there will be only 5 combinations of rotations in total since if excluding the white face, which is tag6 or tag12, there will be only 5 pairs of tags. However, due to time constraints, the functionality of changing orientation was not able to be fully implemented. Now, this functionality can only offset the rotation on the Z-axis with no further calculation on rotating the block to make the white face upward.

In the second half of each loop, the locations of each dynamic block are predicted firstly. In the first half of each loop, the program gets the locations of each dynamic block when the arm starts grabbing and end grabbing the static block. The time between the two observations is also calculated in the first half. At the beginning of the second half of each loop, the two observations are converted from the camera frame to the robot base frame using the *transform_tag_from_cam_frame_to_robot_frame(tags_dict_in_camera, camera_in_robot_base)* function mentioned above to get homogeneous transformation matrices of each dynamic block in the two observations. Then, since the rotating turntable is located at the world frame origin, to prepare for further manipulation and transformation from Cartesian coordinate system to polar coordinate system, the homogeneous transformation matrices of each dynamic block in the two observations should be obtained. The function *find_dyn_and_robot_base_to_world(tags_dict_in_robot_base, robot_base_in_world)* is defined to accomplish this task. This function takes in a dictionary of blocks with homogeneous transformation matrices of each block in the robot arm frame and a homogeneous transformation matrix of the robot arm in the world frame. Then, it transforms each homogeneous transformation matrix of each block from robot arm frame to world frame as return values in the dictionary by using the following equation:

$$H_{Tag}^{World} = H_{Robot\ Base}^{World} H_{tag}^{Robot\ Base}$$

The next step in the second half is to pair up tags in the two observations since both *detector.get_detections()* API and dictionary return value in arbitrary order. The pairing up is executed according to three considerations, which are quadrant, tag number, and value of the angle in between. In the pairing up process, firstly, the program runs a nested loop through the two dictionaries and finds the pairs that the first element of key, which is the tag number, is the same. Then, a polar coordinate system is set for further calculation and filtering. The reference point of the polar coordinate system is the world frame origin. The reference direction is the positive Y-axis of the world frame. The x and y Cartesian coordinates of each paired point are passed in a function called *determine_quadrant_index(x, y)*, which returns the index of a quadrant based on input x and y Cartesian coordinates. The pairs that are not within

one quadrant, such as quadrant 1 and quadrant 3, are filtered out since dynamic blocks are unlikely to pass two or more quadrants during the time of grabbing a static object.

Then, inside the nested loop, the angle and distance of each paired point in the polar coordinate system are calculated by using the following equations:

$$\theta = \tan^{-1}(\frac{y}{x})$$

$$r = \sqrt{x^2 + y^2}$$

The final filter is the differences between angles in pairs. Since the turntable is rotated in a positive direction, the differences in angles that are smaller than 0 will be filtered out. Also, by empirical testing, during the time of grabbing a static object, dynamic objects are unlikely to rotate more than 0.18 radian. Therefore, the differences in angles that are larger than 0.18 radian will also be filtered out.

After pairing is finished, the location of each paired tag is predicted by using the differences in angles calculated previously and the time for grabbing a static object. Firstly, the angular speed can be calculated using the two values mentioned above and the following equation:

$$\omega = \frac{\Delta\theta}{\Delta t}$$

Then, assuming grabbing a dynamic block takes the same amount of time as grabbing a static block, the future rotation of each dynamic block when the grabber under the robot arm reaches the turntable can be predicted by multiplying the time for grabbing a static object back. By summing up the process, it is essentially just rotating the same difference in angles for one more time. Therefore, the angle can be predicted by adding each difference in angles to the respective angle in the second observation.

The predicted x and y Cartesian coordinates can be calculated using the equations below:

$$x = r * \cos\theta_{pred}$$

$$y = r * \sin\theta_{pred}$$

After calculating the predicted x and y coordinates for each dynamic block, the distances between each dynamic block and the robot base are calculated, and the nearest one is selected as the target dynamic block. The new predicted x and y coordinates are then filled back to each homogeneous transformation matrix, and the configuration is calculated using *inverse(value, seed)*. The rest steps are like the ending of the first half. The robot arm firstly moves to the neutral position, then moves to the target dynamic block configuration, grabs the block, and puts it at the current goal location. The only difference is the Z-axis value of the coordinate of the current goal location is slightly increased so that the dynamic block can stack on top of the static block that the arm placed during the first half of the loop.

Evaluation

In order to conduct thorough testing on our experimental approaches, we are expected to design a comprehensive evaluation plan as well as testing procedures. Generally, our group would like to construct the whole tests with two parts. The first is the software-wise testing, which refers to the simulation environment conducted by PC, and the other one is the hardware-wise testing, which refers to the physical manipulation/testing in the Robot Lab. Since most of our project development and testing takes place in

the software simulation, and time we could spend in the real lab is much less, we would estimate a general time distribution for each type of tests and schedule evaluation plans in such a manner.

Based on the previous lab developing experience, we would expect that the software simulation to focus more on an ideal preview of our approaches. It provides visual guidance of what we could achieve in each sub task and how we want to integrate them in a relatively ideal environment. However, since it is free from real-world factors, for instance, the different sources of noise, it is more suitable for the preliminary testing. The physical testing, however, is the way that we could find the gap between the ideal simulation and the real-world testing. It provides precise feedback of our methodology in a more complicated testing scenario. Thus, we could generate new ideas to improve our approach and adjust accordingly. Due to the matter of safety, the software simulation should be thoroughly conducted and checked before we move to the physical testing in the lab.

Software Simulation

When we test and implement our code in the software simulation, we follow a strict guideline, which is that we break up the entire simulation tasks into several sub tasks, which includes: the static block detection and transition, the dynamic block detection and transition, block orientation and block stacking. We will conduct the simulation step by step through each of the sub tasks, make sure each task is successfully implemented and then integrate and test the whole system. Below is the table of success metrics we created to evaluate the progress/status of the tasks and different parameters we applied for judgements.

Success Metrics	Parameters
Is the task completed?	Task completed/incomplete
Is the outcome precise as expected?	Yes/NO
Is the whole process kept under 3 min?	ROS time consumed
Are there any obvious weaknesses?	Weaknesses in the approach

Table 2.1 Success Metrics for simulation tasks

During the evaluation procedure for each sub task, we also designed several milestones for us to keep track of complete status of each component. At each milestone check point, we used the print command to print out important variables, including but not limited to the arm configuration, the path planned for the robot arm, the block matrix, etc. The advantage of such checking method is that we could easily locate where the bug is and fix it whenever there is something wrong in our code.

If we finish conducting all three subtasks and pass all of them, now we should be ready for some more complicated system tests. Since each of the sub tasks is considered as a ‘linear’ operation, we could then implement and combine each task linearly to construct an integrated system for testing. There are multiple test setups we can change to test the system performance under various testing environments. For example, we can try to start the test from both red and blue sides to see if it makes any difference in our approach and arm performance. We could also try different combinations of operations by rotating the sequence of each task and different scoring strategies. Figure 2.2 and 2.3 shows two possible scoring combinations that could be achieved by altering the operations.

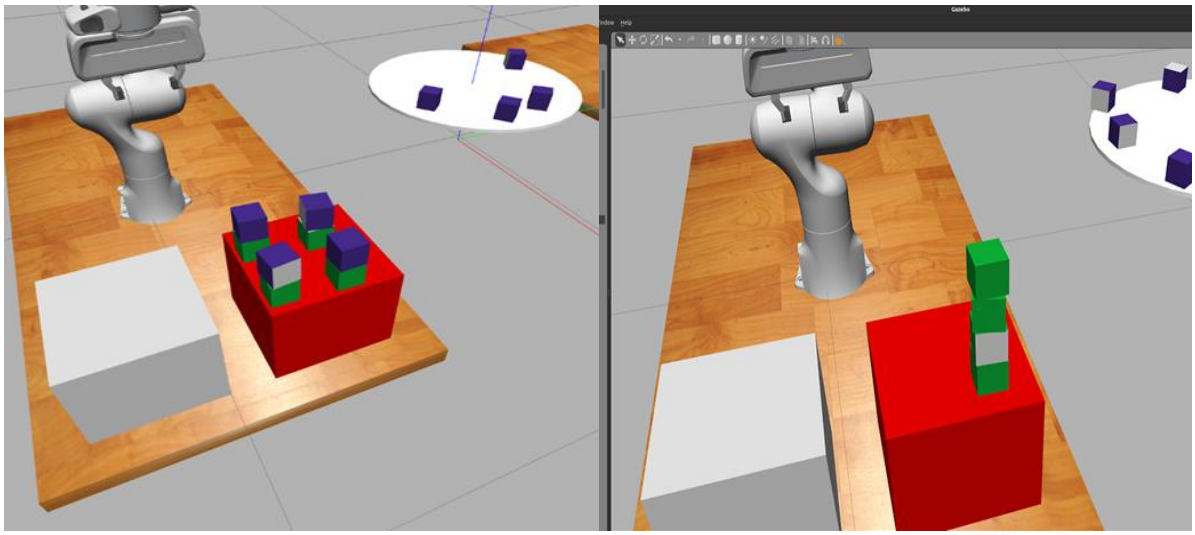

Figure 2.2 (Left): Static Block Transition + Dynamic Block Transition + Block Stacking
Figure 2.3 (Right): Static Block Transition + Block Stacking

Throughout various test setups, we would also record some important parameters in the simulation for future analysis and improvements. The ROS time is an essential part of that since once we obtained the time consumed for different sub tasks, we could have a general idea of what we could possibly achieve in a 3-min competition and work out a more efficient scoring strategy. The success rate is another point we are concerned about because the lower success rate is, the higher possibilities that the system will fail in the physical testing section and in the real competition. We would take both sides into consideration and achieve the balance between the final score and total success rate.

Physical Testing

After conducting the simulation, we are ready to test out our approach in the lab. The testing strategy is very similar to what we applied in the software simulation. We tested the functionality of each sub task first to check if there's any differences between the simulation, and then conduct it into more integrated system operations.

Since there's no need to test each subcomponent in the sub tasks, we generally divided the task into three parts: Static Block transition, Dynamic Block transition and block rotation. For each test category we set a final check point where we determine the status of implementation and how much work still must be done. Table 2.4 shows the brief result of what we achieved in the physical tests. In general, our group's approach passed the sub tasks in the static block part with a high success rate, however, we could not implement the dynamic block transition as what we expected in the software simulation, and more work needs to be done in the block rotation part as well before it could finally be implemented in the real lab. A more detailed discussion regarding the experimental result will be mentioned in the later sections (Refer to Result/Analysis section).

Test Categories	Check Marks
Static Block Transition	Yes
Dynamic Block Transition	No

Block Rotation	—

Table 2.4 Physical Testing check marks

Experimental Result

According to the evaluation plan, here are some of the simulation and testing results we obtained. It is obvious from the result that the real testing environment is not exactly the same as what we expected in our simulation. Consequently, discrepancy has been highly expected, and indeed existed, between the software simulation and the physical testing. To conclude our result more precisely and bring into correspondence with the designated simulation/tests we conducted in the previous section, we will again break up each sub task into some subcomponents and record the implementing status.

Simulation Result

Sub task: Static Block Transition

Overall Implement Status: > 96%

Components of Sub task	Status (complete/not complete) / Success Rate
Static Block matrix in robot base frame	Completed
Static Block Grabbing	96% Success Rate
Static Block Placing in goal position	96% Success Rate

Table 2.5 Simulation Result of Static Block Transition

The static block transition is the content we initially focused on according to our scoring strategy. We conducted the simulation to test the transition performance with various setups, and the overall performance of our approach is quite satisfying. We successfully transformed the static block matrix into the robot base frame, and the success rate of static block grabbing/placing in desired position reached 96%, which made us confident of achieving the full base score of this section in the final competition.

Sub task: Block Rotation

Overall Implement Status: 1/3 completed, need more progress

Components of Sub task	Status (complete/not complete) / Success Rate
Convert rotation representation to quaternion	Completed
Narrow down possible rotation combination	Need Implement
Select correct rotation combo and rotate tag up	Need Implement

Table 2.6 Simulation Result of Block Rotation

The detailed algorithm has been discussed in the methodology section at the beginning of the report. Briefly, we would like to convert the representation of rotation to quaternion, then based on the location of the white surface tag, we can narrow down the potential rotation combinations to a number of

5. Then we can choose different rotation combinations for different scenarios till the white tag is facing up. At the time of competition, we have implemented the conversion of representation and tested it in the simulation, however, we did not go further ahead to explore such an approach to narrow down the rotation combinations and select the correct combinations. This might have something to do with the scoring strategy we set up at the beginning, which is that rotation has a relatively low priority among different tasks, but this part is absolutely worth more development.

Sub task: Dynamic Block Transition

Overall Implement Status: 3/3 completed, one with low success rate

Components of Sub task	**Status (complete/not complete) / Success Rate**
Obtain 2 frames of Observations	Completed
Cartesian/polar Coordinate Conversion	Completed
Dynamic Block Grabbing and Stacking	18% Success Rate

Table 2.7 Simulation Result of Dynamic Block Transition

The subcomponents we designed for testing the dynamic block transition are to capture the two frames of the observation so that we could work out the accurate angular speed of the platform, to convert between Cartesian and Polar Coordinate system, and to grab and stack the dynamic block onto the corresponding position, which in our case on the top of another static box. Figure 2.8 and 2.9 shows the process of grabbing the dynamic block and the outcome of the stacks of one static block and one Dynamic block.

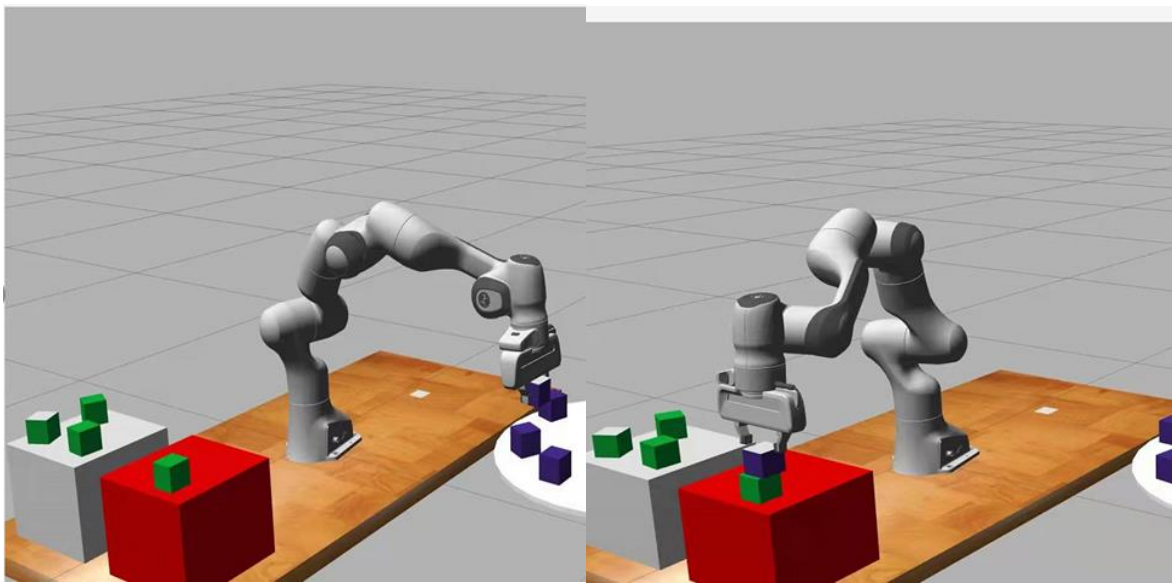

Figure 2.8 and 2.9 Processes of grabbing and stacking dynamic block onto static block

With the help of the visual system in the simulation, we successfully implemented the first two subcomponents. We observed that when the system finished capturing the frames and converting the coordinate system, it would still have a relatively low success rate (18%) when grabbing and stacking the dynamic block. We also notice that at most times when failure happens, the main problem is at the

detection of the dynamic block as well as the grabbing process. Based on this observation, our guess for one potential reason for such a problem is that our estimation of the initial position of the platform is not quite accurate, and also the angular speed we calculated is not precise enough, which would result in a failure of the detection and grabbing of the dynamic block. The physical testing in the lab verifies that our approach has a low overall success rate for the dynamic transition and stacking part, which we will go over in the next section.

Physical Testing Result

Table 2.10 demonstrates the experimental testing results in the lab. Overall, the result is consistent with what we expect according to the performance of our simulation.

Components of Sub task	**Status (complete/not complete) / Success Rate**
Static Block Transition	Completed
Dynamic Block Transition	Need Implement
Block Rotation	Need Implement
Block Stacking	Completed

Table 2.10 Experimental Testing Result

As a result, we could successfully fulfill the requirements of static block transition and stacking, which matches what we anticipated from the simulation performance. For the dynamic block transition part, we obtained a success rate of 18% in the simulation and when testing in the real lab, but we could not implement it for almost all time. Besides the approach to wipe out the perception noise that may cause the discrepancy between the software simulation and the physical testing, we also need to keep exploring a more reliable approach in the simulation for dynamic block detection and grabbing to increase its success rate.

We also record the total ROS time used for the whole process. Figure 2.11 shows the process of operation we record in the lab. We tested with the example of moving all 4 static blocks into the desired position without rotation and stacking, which is one of the most basic manipulations in this project, and recorded the time consumed.

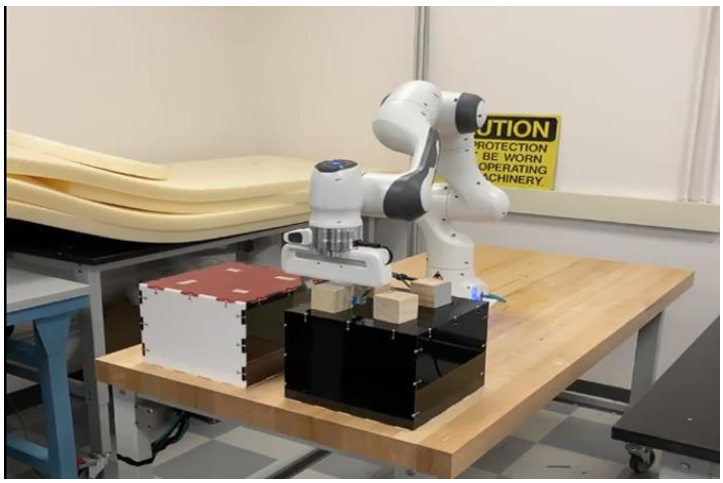

Figure 2.11 Time Test (4 Blocks)

Table 2.12 shows the total time it takes to complete such operation in both software simulation and the physical testing. By the data we collect, the time it takes for our approach to be fully implemented in the physical testing is even less than that in the simulation. Due to the limitation of time, we only had a chance to test out this configuration. We would like to conduct more tests and try multiple combinations and see what is the most efficient strategy that we could score most points in a limited time period.

Tests	Total Time (s)	Time/Block (s)
Simulation	128	32
Physical	105	26.25

Table 2.12 Time spent in simulation/physical testing

Analysis

Our strategy for the static blocks in simulation transferred very well to hardware experiments. Because our algorithm only needs one instance of tag detection at the beginning to find the static blocks, we avoid perception challenges such as occlusions and accumulating sensor error. However, we were not able to fully implement successful solutions for block rotations and dynamic block pick-and-place due to time constraints.

For block rotations, our remaining challenge is selecting the correct rotation combination in order to rotate the block to the desired orientation and converting it into commands for the robot gripper. We are able to calculate the offset from the z-axis of the static block, but we still need to calculate which rotations about the x-axis are required to rotate it. Then, our next step would be to command the gripper to pick it up and rotate it by 90 degrees until it is in the desired orientation with the white tag facing up. We would be able to check if this task is completed successfully by checking the visible tag with the tag detector.

Dynamic block pick-and-place had some chance of success in simulation, around 20%. However, our strategy would only work in simulation because it relied on using two instances of block detection to predict the block trajectory. Our perception system would only be able to give us one instance of the initial block positions, so we would have to change our strategy for the hardware implementation. One potential approach for the hardware implementation would be to experimentally measure how quickly the turntable spins and use the angular velocity to predict the trajectory of the actual block. Once we can make an approximation of where the block would be at a specific time, we can position the gripper in the path of the block before it gets there. Then, we can use proprioceptive feedback from the joint torques to determine if the block has made contact with the gripper and grasp it. This approach would be fairly robust to error in the block trajectory prediction since there is some buffer (placing the gripper ahead of the block) from using the much more accurate proprioceptive feedback to detect the block.

Lessons Learned

From this project, we see that open loop interaction with mobile objects is more demanding than handling static targets, whereas the integration of close-loop sensors shall reduce the uncertainty and improve the performance. Some tasks (such as orienting the blocks) might not be realizable in a single operation by the robot arm, but rather needs to be completed in discrete steps (flip the block stepwise). Importantly, we have learned how we can apply Robotics kinematics and planning to build solutions in a complex environment, and how to accommodate the gaps between simulation and physical testing.

The static block task works well in simulation and transfers to reality with no problem. We use a predefined Z axis coordinate. Therefore, the inadequate Z axis perception in the vision system did not have much interference when locating the blocks. The dynamic block task does not work really well in simulation. This is probably because our assumption, which is the grabbing time for all blocks should be similar, is not accurate, which results in our prediction in angle rotated is not accurate. Also, the dynamic block task cannot transfer to a physical robot since the vision system in the lab environment changed to only one perception at the beginning of the program, and we did not have enough time to adapt this adjustment.

References

MEAM520. (n.d.). Final Project: Pick and Place Challenge. Canvas. Retrieved December 12, 2021, from https://canvas.upenn.edu/courses/1607214/files/folder/Final%20Project?preview=104085771.

Spong, M. W., Hutchinson, S., & Vidyasager, M. (2006). Robot Modeling and control. John Wiley & Sons.