MEAM520: Final Project Report

Gaurav Kuppa, Evan Grant, Joanna Wang, Jason Li

December 15, 2021

Project Definition:

The MEAM520 final project was a "Pick and Place Challenge". Our task for this project was to reliably grasp both stationary or mobile blocks and stack them on a goal platform, right-side up, and in as tall a tower as possible.

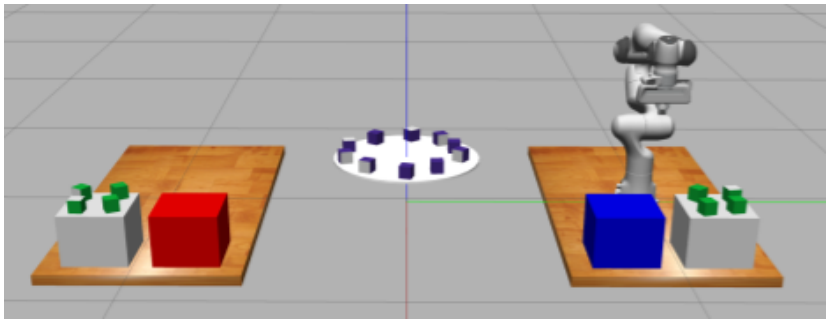

Figure 1: Competition Environment

The competition occurs in a shared environment between two teams. Each team has access to two platforms in front of the arm, one for the static blocks and one for the goal platform. The four static blocks are randomly dispersed on a stationary platform while the dynamic blocks are placed around the edge of a rotating turntable. Both teams share a single dynamic block turntable and the pool of dynamic blocks. Scoring is based on the type of block placed, how high it is stacked and its orientation through the following equation:

$$Points = Value \times (Altitude + SideBonus)$$

Value *= 1 for static, 2 for dynamic*

Altitude *= distance from the center of the object to the surface of the goal platform in millimeters*

SideBonus *= 50 if white side of object is pointing upward*

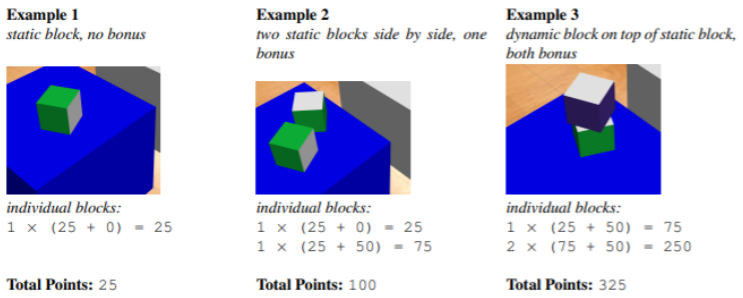

Figure 2: Scoring Example

We are given 3 minutes to score as many points as possible through any combination of altitude, SideBonus and type of block. Our strategy will have to weigh the value of each block along with the time cost of each action. We will also have to take into account our opponents strategy as both teams share a common dynamic block pool.

Approach:

Our overarching strategy was to quickly stack all the static blocks first while correctly orienting some of them, then stack as many dynamic blocks as possible in the remaining amount of time. For static blocks, rotating upside down blocks takes a significant amount of time that takes away from our ability to retrieve dynamic blocks. Correct orienting all static blocks results in a reliably average points total while dynamic blocks could lead to a higher scoring potential while being less reliable. By stacking dynamic blocks last, we also leave the turntable open to sabotage or early picking of dynamic blocks by the opponent team. However, by stacking dynamic blocks on top of an existing tower, we gain a large altitude bonus which offsets the risk.

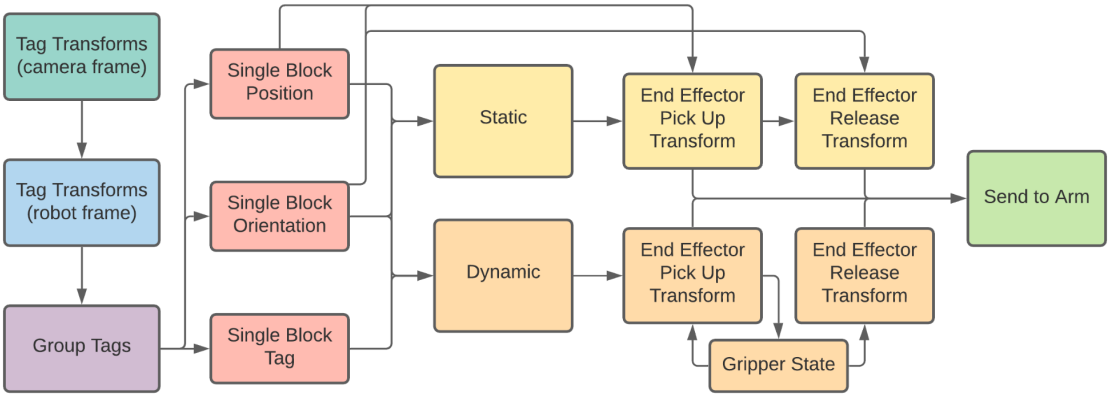

Figure 3: Planning and Control Architecture

To accomplish this, we determined several sub-components to solve. First, was to create the transform from block to robot frame, depending on which team was chosen. Next is to determine block pose. After we determine block pose, we choose the order to pickup detected blocks based on our strategy. If the chosen block is a static block, then we compute the required end effector transform to pick it up and the required rotation to correctly orient it. If the next block is dynamic, we command the arm over the table and repeatedly open and close the gripper until it detects a block is successfully grasped. All blocks are then stacked in a single tower.

Method:

Transform:

With the assumption that we are using a real-life visual system, we were expecting to receive camera odometry information of the tags. In other words, we would receive tag odometry in the camera frame. In order for the Panda arm to manipulate these blocks, it needs to know the locations of the tags in the robot frame. Given the relative position of the robot frame and tag 0 frame and the robot and the world frame, we were able to construct the following transformation matrices:

$$\mathrm{T}_{world}^{robot} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.978 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathrm{T}_{tag0}^{world} = \begin{bmatrix} 1 & 0 & 0 & 0.500 \\ 0 & 1 & 0 & -0.978 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

With that, the formula to find the tags in the robot frame is as follows:

$$\mathrm{T}_{tag}^{robot} = T_{world}^{robot} * T_{tag0}^{world} * T_{camera}^{tag0} * T_{tag}^{camera}$$

Block Detection:

Following the initial transform, we needed to convert transformed tag input into block pose information which would better suit the pick and place task. Block detection was first developed with the expectation of tag input data from a computer vision system. Thus our initial approach included signal conditioning steps - primarily software filtering to reduce noise by averaging data from several successive frames and constraining detections to known altitudes (typically 0.20m for blocks on any playable surface). However, after vision detection became infeasible, we transitioned to an approach which assumed perfect initial tag detection for static blocks. We also assumed no viable data for dynamic blocks.

With valid transformed tag frame data, we then established a pipeline to arrive at block poses. First, we developed a method to find tag-to-block correspondences. This involves an “inset tag” step in which every detected tag frame is translated 0.025m in its respective -z direction. Because the z-axis of every tag points outward from the block center, and given block width of 0.05m, this step guarantees tags are shifted toward the center of the block to which they belong. Any tags which remain within a distance of 0.05m of other tags are considered belonging to the same block and are then grouped and appended to a list containing tag entries. The number of resulting tag groups indicates how many distinct blocks are detected. Note: during simulation testing, we found that multiple tag detections for static blocks never occured because of the overhead angle of the virtual camera - however, this step was preserved to flexibly handle any scenarios when interfacing with the real system and instructor-generated tag data.

In the next step, we leveraged the known transforms from block frame to that of each numbered tag by converting the first tag in every block group to the frame of its block. This step outputs a single accurate block pose (4x4 Transformation matrix) for each detected block.

The final block detection step simply sorts blocks into two lists - Static and Dynamic - after reading the names for each tag detected for a given block, where tags 1-6 correspond to static blocks and 7-12 to dynamic.

Static Blocks:

The static blocks are randomly dispersed and oriented on a stationary platform. We need to be able to pick up the static blocks at any orientation and location while also rotating the blocks to their correct orientation before stacking them. To accomplish this, we first need to determine which orientation the block is in for the pickup phase. We distinguished four different orientation cases for the static blocks. In all pickup cases, the end effector is always facing straight down, and we rotate the end effector around its z-axis to align it with the static blocks.

The first case with the block being correctly orientated to begin with is the simplest. We align the end effector +y-axis and +x-axis with the block -y-axis and -z-axis respectively to create a transform to grab this block. The second case is with the block facing straight down. In this case, the end effector coordinate frame is parallel with the block coordinate frame. The third and fourth cases concern blocks on their side. If the block y-axis is parallel to the world z-axis, we keep the end-effector +x-axis along the block -z-axis. The end effector +y-axis is along the block ±x-axis with the sign depending on whether the block y-axis is pointing along the world positive or negative z-axis. Similarly for the case where the block x-axis is

vertical, end effector +x-axis is always aligned with block -z-axis while the direction end-effector +y axis is aligned with is dependent on whether block x-axis is up or down.

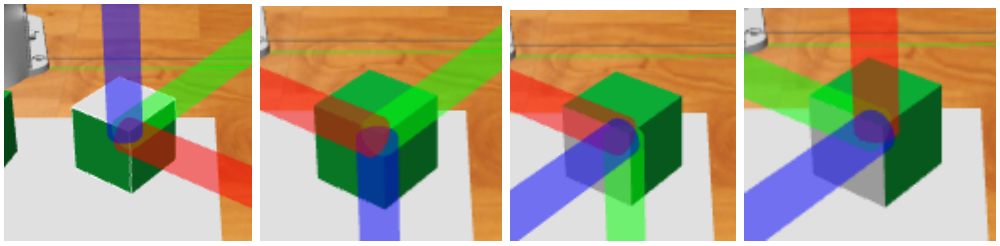

Figure 4: Different static block cases

After determining the correct end-effector transform to grip each block, we use our IK solver to generate the correct joint configuration to achieve those poses. We solve for one pose at the block to grip it and one pose 10 cm above the block as a neutral position. We command the Panda arm to the pose above the block, then to the pose at the block to pick it up, before returning to the neutral pose above the block. This concludes the pickup phase.

For stacking blocks at the goal, we change the end-effector orientation to result in different rotations. For blocks with the white side already facing up, we simply grab and place the block without rotation. For blocks on their side, we perform a single 90 degree rotation to correctly orient them as demonstrated in the following figure. This pose was chosen as it gives us the most clearance between the end-effector and the goal table.

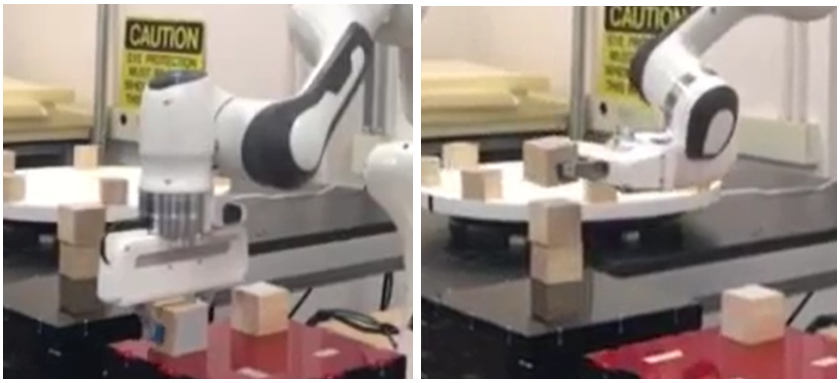

Figure 5: Sideways Block Reorientation

For upside down blocks, we require a two step process. The first step is to pick up and rotate the block 90 degrees to an intermediate staging location. At this point, the upside down block is now simply treated as a sideways block and we stack it after performing another 90 degree rotation.

Figure 6: Upside Down Block 2-step Process

To determine the required joint configurations to stack the blocks, we look up the correct values from a precalculated dictionary. Since we only place the blocks at one location and in only two final end-effector orientations, there are a finite number of possible goal poses. This way, we do not have to rerun our IK solver and can simply look up the correct values.

Dynamic Blocks:

With the camera vision system no longer being implemented, we could not track the locations of the dynamic blocks in real time anymore. However knowing the spacing and orientation of the dynamic blocks, as well as an estimation of the turntable's angular velocity we can find a pose where the gripper will eventually encounter a block.

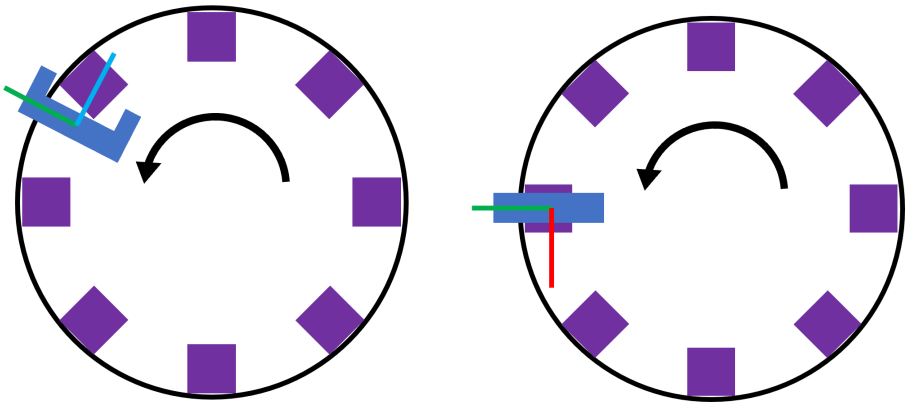

Figure 7: a) initial dynamic approach. b) final dynamic approach

We initially envisioned this strategy as holding the gripper parallel to the turntable, so as the blocks rotated, a block would eventually be "caught" in the gripper. By holding the gripper in this position for a certain amount of time (if no blocks have been removed) then the gripper is guaranteed to have a block, and we can command the gripper to close and move the block to our goal. With this approach, we had difficulty getting our IK to solve for a position where the gripper was horizontal, even by feeding in seeds

that we knew were almost exactly there. This most likely was a result of how we implemented task prioritization in our solveIK.py function. We manually determined joint angles that got us close to the desired position to test it, but we found that the gripper was colliding with the turntable in simulation.

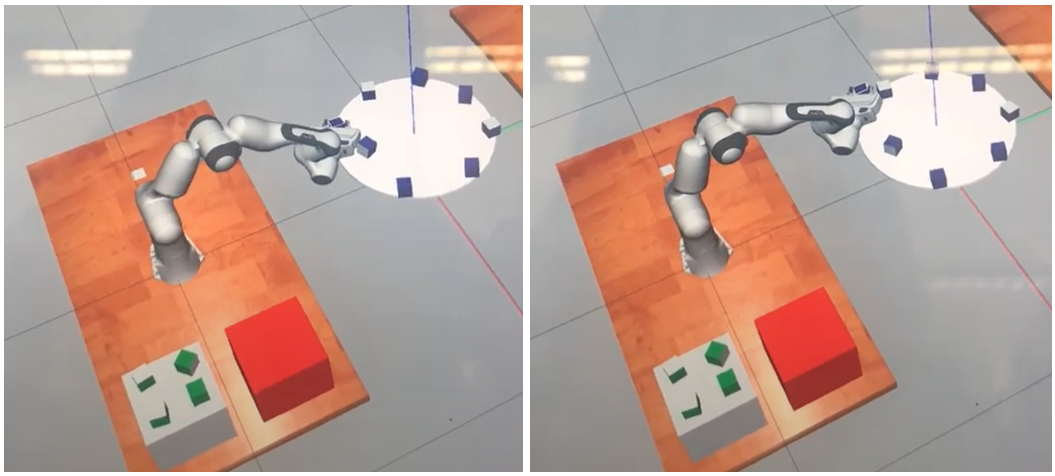

Figure 8: Horizontal dynamic block approach picking up a block but colliding with turntable

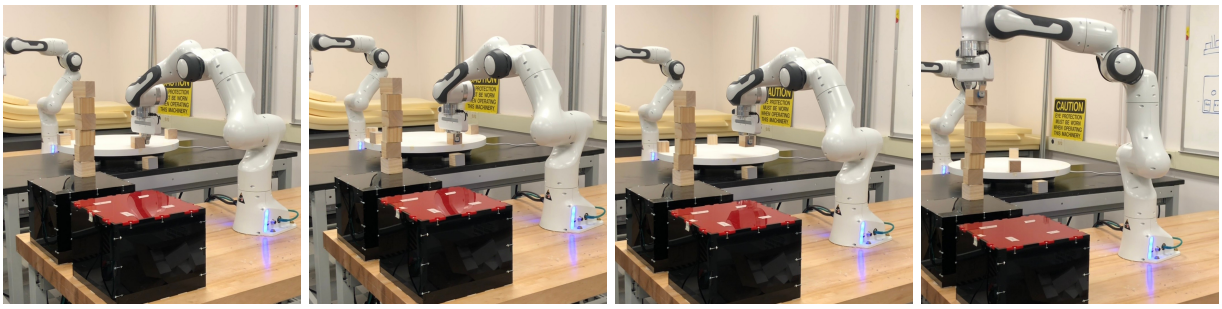

Figure 9: Dynamic Block Approach

Due to the difficulty with solving the inverse kinematics and the potential for collision, we changed our approach to holding the gripper vertically above the turntable such that the blocks would travel between the two grippers. The gripper would periodically open and close and based on gripper positions we could determine if a block was grasped. Once a block is grasped, the arm will move to the goal position. The end effector location was determined based on the environment parameters given to us (x = 0, y = ±0.698, z = .225) so that it was centered on the block position closest to the arm. Using solveIK.py with a seed where the arm was already turned in that direction, we got the joint angles which we used directly. Similar to the static blocks, we designated intermediate arm positions, 0.1 m above the pickup locations, to prevent the gripper from unintentionally colliding with any blocks. One thing we needed to tune was the time between opening and closing the gripper. Ideally we want the gripper to open and close in the time it takes half a block to rotate past the pickup point.

Offline Computation:

Following extensive preliminary testing with 15 trials in simulation, we discovered our solution performed well on the metrics of achieving a 4-block static tower (our standard stacking configuration), full SideBonus, and zero collisions with blocks or the surrounding environment. However, the solution proved to be computationally inefficient, and as a result we failed to achieve a 4-block stacking time under 140 seconds. In fact, in the majority of trials - and all trials with even one facedown block requiring SideBonus reorientation - we exceeded the 180 second time allotment for the competition.

To address our slow performance, we evaluated four solutions: adjusting learning rate and centering task parameters, seed fine-tuning, IK velocity solving, and offline IK computation. Adjusting the IK solver parameters did little to help with convergence time. Seed tuning was performed constantly - but never resulted in a perfectly stable or efficient solution. The IK velocity solution (discussed in detail in our *Results and Discussion)* was ultimately rejected for safety reasons and because offline computation proved a promising method which could also preserve our existing methodology.

During the aforementioned trials, IK solve time was output for each step in the pick and place task. For each block, up to 1.5 seconds were spent solving the IK problem required to stack it. We then realized, because our environment is well defined and largely predictable (one of the reasons why we opted for no path planning algorithm such as RRT*), the IK solution needs not be computed live for every waypoint. The only instances in which a pre-computed IK solution is not viable is when the arm moves to pick up a block with an unpredictable initial pose. We thus developed a robust method for offline IK computation for all other waypoints which could be stored and referenced by our program each time a path was constructed.

The exact methodology for this offline step required developing a dictionary-building class in python. It was designed such that all priors like block width and desired stack location could be input and the code would automatically generate a dictionary for both Red and Blue teams giving guaranteed successful IK solutions for all waypoints.

Throughout preliminary testing and by constantly seed-tuning during our process, we generated a list of potentially viable seeds which could be utilized by this solver. Because the time to converge to a solution was of no consequence during offline computation, we employed a “brute force” approach by feeding the program the full list of seeds to iterate through in the search for a solution. If a successful solution using pre-defined seeds could not be found, randomized seeds were then generated from a uniform distribution

within joint limit bounds and used until a solution was found. We found the randomized seeds were often only necessary for some of the more exotic waypoints such as our facedown block staging area or stack height positions above 6-7 blocks. Despite an inconsistent runtime, this method arrived at successful solutions for every waypoint 100% of the time.

Testing and Evaluation:

Simulation Trials (before offline):

Table 1: Preliminary Testing Static Tower Stacking

Trial	Time (s)	Successful Stack?	Joint Limit Errors?	Block1 Move Time (s)	Block2 Move Time (s)	Block3 Move Time (s)	Block4 Move Time (s)
1	171.576	Y	Y	N/A	N/A	N/A	N/A
2	194.944	Y	N	N/A	N/A	N/A	N/A
3	196.86	Y	N	80.70	39.17	39.82	37.17
4	147.90	Y	Y	35.36	35.07	32.40	45.07
5	264.44	Y	Y	50.74	84.54	44.94	84.22
6	246.433	Y	Y	40.08	47.27	46.66	91.59
7	237.259	Y	Y	87.17	38.00	37.56	44.50
8	147.548	Y	N	35.30	37.93	37.74	36.58
9	195.894	Y	N	34.85	35.23	88.19	37.62
10	144.133	Y	N	32.47	35.14	37.57	38.95
11	190.166	Y	N	34.76	81.03	37.32	37.05
12	185.044	Y	N	33.18	33.03	73.79	40.04
13	173.82	Y	N	71.76	30.12	33.28	33.66
14	144.431	Y	N	34.72	34.32	38.16	32.21
15	184.536	Y	N	31.25	37.06	37.70	73.52

Table 1 gives the results of preliminary testing after implementation of a working static tower stacking solution, prior to development of a dynamic approach or offline computation. As mentioned previously, the majority of these trials exceeded the 180 second time allotment just to stack 4 static blocks in a single tower.

Important aspects to note:

- for blocks taking approximately 70-90 seconds to complete moving, these are facedown blocks requiring reorientation to achieve Sidebonus.
- The column “Joint Limit Errors?” reports the persistent IK solver failure which we could not ultimately manage to address with adjusting learning rate and centering task parameters, seed fine-tuning, IK velocity solving, or offline IK computation. Planning a path for these block pick and place tasks take longer as we pass a list of seeds to the solver.
- In ***every*** case, joint limit errors were the result of J6 rotation, and the arm controller would simply constrain that joint to its limit and continue with the pickup task. These errors were often not far outside joint limits.
- We experienced a 100% success rate at a 4x block stack with full SideBonus

Simulation Trials (after offline):

Table 2: Final Testing in Simulation

Trial	Time (s)	Successful Stack?	Joint Limit Errors?	Block1 Move Time (s)	Block2 Move Time (s)	Block3 Move Time (s)	Block4 Move Time (s)
1	112.518	Y	N	25.53	24.85	27.41	33.84
2	119.306	Y	N	28.07	31.26	28.28	30.8
3	198.01	Y	Y	62.42	50.35	29.48	54.875
4	150.616	Y	Y	45.14	34.51	30.5	39.57
5	162.045	Y	N	67.94	30.51	34.77	27.93

Table 2 gives a few trials of static tower stacking in simulation following optimizations including offline computation and a code review to remove unnecessary or blocking code. In the first two trials, we experienced no joint limit errors or facedown blocks, and were able to complete the 4x stack in under 2 minutes for an average improvement over preliminary testing greater than 30 seconds. However, we were unable to solve the IK failure issue, and in trials 3 and 4, multiple failures and facedown blocks resulted in reduced performance. Failing IK cost more time in these trials than in the preliminary trials because we entered a longer list of seeds and attempted to alter grip target position following several failures to converge.

Trial 6 highlights a dramatic improvement in reorienting facedown blocks. A task which previously took on average 80 seconds per block now takes less than 70. This trial was repeated but not recorded here for the sake of brevity. The reason for this improvement is a direct result of pre-computed IK solutions from the offline step, as facedown block paths required 2X the number of IK solutions.

Results and Discussion:

IK velocity:

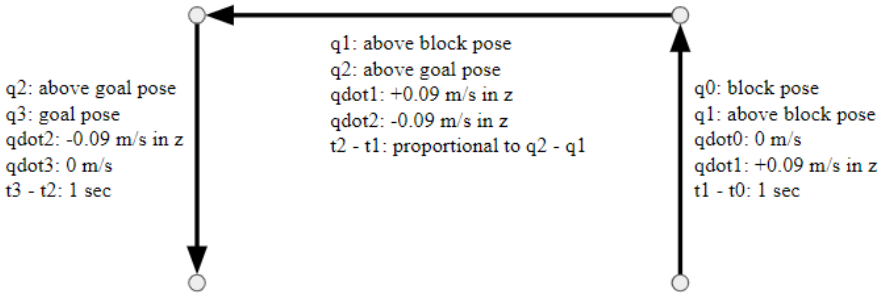

Figure 10: Cubic Polynomial Trajectory

One major source of time delays are the pauses between safe_move_to_position() commands. Using this function, the arm moves from point to point while starting and ending with zero velocity. We could potentially save a lot of time if we can chain points together into one continuous trajectory. To accomplish this, we solve a chain of cubic polynomial trajectories through each point where the final conditions of one path become the initial condition of the following path.

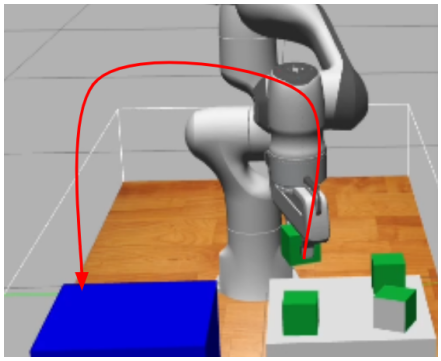

Figure 11: Continuous Path using Cubic Trajectory (https://youtu.be/x4s_mVlhGT0)

Using these parameters, we solve the cubic polynomial giving us the coefficients to a parametric equation that results in the joint position and joint velocity with respect to time. The joint velocities were solved using velocityIK. Using this equation, we input ROS time to generate the desired position and velocity needed to command the arm through safe_set_joint_positions_velocities(q, qd).

Static Blocks:

After optimizations, our static block stacking solution performed exceedingly well. Prior to the classwide demonstration, we also programmed functionality which could enable us to turn off facedown block reorientation and place those blocks without SideBonus first at the bottom of the stack. With reorientation

off, we could possibly stack all static blocks in approximately 113 seconds quite reliably - allowing us more time to collect the less reliable, yet highly valuable dynamic blocks.

Competition strategy aside, we met our goal of reliable full static stacking under 180 seconds. However, the intermittent and indeterministic failure of our IK solver continued to present a challenge throughout the entirety of our development. If we were to take this project further, our first task would be to isolate the failure and mitigate it however possible. It is likely that further refinement of our seed would produce more robust results - and we might even utilize the seed searching functionality of the offline configuration generator to accomplish this.

Dynamic Blocks:

Due to very slow simulation, we have limited results for dynamic block attempts on the robot arm. We only have one successful attempt at dynamic blocks in testing. Of the 4 dynamic blocks we attempted to pick up, we were able to add 3 to our static block stack, and knock one off the turntable. The success of this trial is due to us tuning the timing and adding a 0.5 second delay between opening and closing the gripper when blindly grabbing blocks. Additionally, the blocks were still regularly arranged on the turntable. In other attempts, the timing of the gripping was too fast and blocks were unable to get between the grippers. Even though we had this adjusted in testing, we were once again gripping too quickly to get any blocks in competition due to other speed changes we made in our code. We could make our approach more reliable with more testing on the robot to adjust this time delay.

Competition:

As the final project deadline approached, we prepared for the final competition on Dec. 10 at Wu & Chen. In the first round, our robot got all of the static blocks, with two static blocks collecting the Side Bonus. We, intentionally, did not attempt to get the Side Bonus for the blocks facing down, as this would add an extra 30 seconds to our time per block. As a part of our game strategy, we wanted to give our robot a chance to get the dynamic blocks. In the first round, we were unable to get the dynamic blocks due to a bug in our code. We commented out the line of code that would cause the robot to continuously open and close its gripper, in anticipation of a dynamic block. In the second round, we faced a similar situation with the static blocks. We had gotten all static blocks, with two collecting the Side Bonus, and we did not collect the Side Bonus for the blocks facing downward. For this trial, we were able to fix the bug for the dynamic blocks from the previous trial, but the robot arm was unable to time its gripper to get a dynamic block due to timing issues. Despite our shortcomings, we moved onto the next round as the first place

team from our division. In the first round of the knockout stage, our robot had severely malfunctioned. In specific, it had exceeded joint limits and refused to move after it had reached this point.