MEAM 520 - Cynthia Sung

Suveer Garg, Vera Lee, Zador Pataki, Ayaaz Versi

Final Project Report

Section 001

Due: 14 December, 2020

This report can be shared with the MEAM 520 Fall 2020 class.

1 Methods

1.1 Controller

1.1.1 Controller Overview

To control the robot for the duration of a competition round we decoupled the problem into three sub tasks, each of which are called in a main controller file. The goal of the first sub-task is to determine which block should be picked next. Using methods outlined in Subsections 1.2.1 and 1.3.1, we calculate the ideal dynamic and ideal static block that should be picked next. If there is an ideal dynamic block to be picked, then the robot will go for the dynamic block; otherwise, it will go for the ideal static block. The second sub-task involves approaching and gripping the block to be picked, with the methods outlined in Subsections 1.3.2-1.3.3 for the dynamic blocks and Subsections 1.2.2-1.2.3 for the static blocks. In the final sub-task, if the robot successfully picked up a block, the robot is commanded to the goal platform and the block is stacked according to the method in Subsection 1.4. These sub-tasks are repeated until there are no more blocks that can be feasibly picked.

1.1.2 Moving Between Configurations and Handling Errors

To move between various configurations, we tune the manner in which the robot is commanded to move on a case-by-case basis. In cases where the robot can move quickly and collisions are not a concern, we call the function `lynx.command()` to directly update the setpoint of the PID controller for each joint, using an appropriate sleep timer. These cases include moving to the dynamic, static and goal platforms, when there are no obstacles in the way, and moving during the gripping and extraction of the blocks on the dynamic platform; when handling dynamic blocks, it is crucial that the approach of the robot is fast (see Section 1.3.2). In cases where we are worried about collisions, we use a function which interpolates between current state and goal state and commands the robot to move along all waypoints with appropriate sleep timers. By setting a larger number of waypoints, the robot moves slower and more accurately. These cases include grabbing the static blocks, where we are worried about knocking them off the static platform, and the stacking of all blocks, where we aim to avoid knocking over the stack.

We encoded a safety mechanism in the controller to rapidly correct our robot in cases when it fails to pick up blocks. In the controller, each function for each sub-task is called independently, and returns a boolean based on whether or not the corresponding task was completed successfully. In the cases where the boolean returns false, the robot restarts the process starting from the decision of which block to pick. This is especially useful when a dynamic block moves out of our robot's reach or is removed by the other team. In such cases, the algorithm immediately reevaluates which block is most optimal to pick up.

1.2 Static Blocks

1.2.1 Deciding Which Static Block to Pick

First, the algorithm isolates all remaining cubes on the static platforms based on their [x,y,z] coordinates and transforms their poses into the robot frame by multiplying by the following homogeneous transformation matrices depending on the robot color:

$$T_0^{\text{robot, blue}} = \begin{bmatrix} -1 & 0 & 0 & 200 \\ 0 & -1 & 0 & 200 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \text{ or } T_0^{\text{robot, red}} = \begin{bmatrix} 1 & 0 & 0 & 200 \\ 0 & 1 & 0 & 200 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

We identified the ideal picking condition for each block to be one where the end-effector picks up the block from the top - in this picking condition, the actuation of joint 5 of the robot allows the gripper to match the orientation of the block exactly. For each of the static blocks remaining on the static platforms, we calculate the homogeneous transformation matrix from the end-effector frame to the robot's base frame where the end-effector would be positioned directly above the cube in consideration, with the gripper orientation matching the cube orientation exactly. This was done by setting the z-axis of the end-effector to be in the -z direction of the robot frame, and setting the y-axis to be equal to one of the axes of the block. The x-axis of the end-effector is then defined as the cross product of the latter two axes. The end-effector position is set to be equal to the [x,y,z] position of the block, and the z position is increased by 40 mm to be above the block. Since the orientation of the x and y axes of the end-effector as described above is not necessarily feasible, we use inverse kinematics and check each consecutive 90 degree rotation about joint 5 for feasibility in configuration.

The decision for which static block to pick first is dependent on the orientation of the +z-axis of the cube and the current stack height on the goal platform. If there are no blocks on the static platform, the ideal block to pick is one where its +z-axis (white face) is aligned with the +z direction in the robot frame, as the robot can stack the block vertically, with its end-effector z-axis pointing downwards. However, if there is at least one block in the goal platform stack, this orientation for stacking becomes infeasible, and the next ideal block to be picked is one where the +z axis of the block (the block's white face) can be oriented in the

same direction as the +y axis of the gripper. This allows the robot to later stack the block with the white side facing up, as outlined in Section 1.4. If no remaining static blocks meet this requirement, then the next most ideal block to be picked is one where the +z axis of the block faces the -y axis of the gripper. Although we would not be able to obtain the side bonus from this gripping position, it still ensures that the robot is safely able to stack the block; since the x-axis of the end-effector corresponds to the location of the gripper fingers, it is ideal to have the +z-axis of the block in the same direction as the y-axis of the end-effector so as to avoid the gripper fingers from knocking over the stacked blocks during stacking.

If the [x,y] coordinate of the block is outside the reachable workspace of the robot's wrist, then we are unable to pick up the block from the top, and our robot will pick it up from the side according to Section 1.2.3. However, we found this condition to be rare in most random block placements, so we did not optimize the function to grip blocks from the side; our robot will only pick these blocks up if no other reachable blocks remain on the platforms.

1.2.2 Approaching and Gripping Static Blocks from the Top

As outlined in Section 1.2.1, to pick a static block the robot will first move to a configuration that places the end-effector 40 mm above the block with the x and y axes matching the orientation of the block. Once the robot reaches this configuration, we lower the gripper down to the block using methods outlined in Section 1.1.2, then set joint 6 to -5 mm, which we experimentally found to be ideal for consistently gripping the block without dropping it. The robot then returns to its initial position 40 mm above the static platform, and we call lynx.get_object_state() to check that the static cube was successfully picked up based on its z-position coordinate. If the static block was successfully picked, then it is stacked according to methods outlined in Section 1.4.

1.2.3 Approaching and Gripping Static Blocks from the Side

Our algorithm first checks all the blocks that cannot be gripped from the top to determine if any of the blocks have their +z-axis aligned with the +z-direction in the world frame - this is the ideal block to pick, since it would allow the robot to stack the block with the white side facing up. Otherwise, it loops through the [x,y,z] position of each block and identifies the closest block in distance from the base of the robot as the block to pick next. We define the normalized vector from the base of the robot to the center of the block as the appropriate direction for the z-axis of the end-effector, set the y-axis of the end-effector facing the +z-direction in the world frame, and calculate the x-axis as the cross product of the two axes. Inverse kinematics for the resulting homogeneous transformation matrix is used to calculate the desired configuration for the robot.

1.3 Dynamic Blocks

1.3.1 Deciding Which Dynamic Block to Pick

The dynamic blocks periodically enter and exit domains in which the end-effector can reach the blocks with a feasible orientation. To tackle this problem, we first had to decide the optimal block orientation for picking, as depending on the orientation, the feasible picking domains were limited differently. As time is a crucial factor, we decided to pick all blocks from the top with the z-axis of the end-effector pointing down. Compared to picking up blocks from the side, our method allows for greater freedom in orientation and does not require much waiting time for blocks to enter feasible domains.

While we experimented with several different methods for choosing the ideal dynamic block (Section 3), our ultimate strategy was to split the dynamic table into ranked domains based on feasibility of picking, and make decisions based on where each block is on the table.

We defined three domains on the dynamic platform: $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ s.t.:

$$\mathcal{D}_1 \succ \mathcal{D}_2 \succ \mathcal{D}_\mathcal{S} \succ \mathcal{D}_3,$$

where $\mathcal{D}_\mathcal{S}$ is the domain on the static table. In this case, if block $b_1 \in \mathcal{D}_1$ and $b_2 \in \mathcal{D}_2$ the robot would choose b_1 as the ideal dynamic block. The final decision criterion is that if the ideal block is in $\mathcal{D}_i$ with $i \in \{1, 2, 3\}$, then pick the block in $\mathcal{D}_i$ closest to the center of the dynamic table.

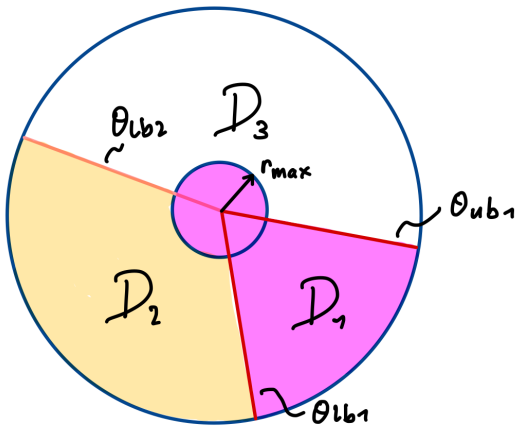

Figure 1: Decision domains for picking dynamic blocks

We defined that domains as follows:

$$\mathcal{D}_1 : \mathcal{D}_\mathcal{D} \wedge (\theta \in [\theta_{lb1}, \theta_{ub1}] \vee r \in [0, r_{\max}]),$$

$$\mathcal{D}_2 : \mathcal{D}_\mathcal{D} \wedge \theta \in [\theta_{lb2}, \theta_{ub1}],$$

and

$$\mathcal{D}_3 : \mathcal{D}_\mathcal{D} \backslash \{\mathcal{D}_1,\ \mathcal{D}_2\},$$

where $\mathcal{D}_\mathcal{D}$ is the domain of the entire dynamic table, θ is the angle of the center of the block relative to the world frame, θ_{lb} and θ_{ub} define the upper and lower angle bounds for each domain, and r is the distance of the cube from the center of the dynamic table. These domains are also presented in Figure 1.The set of parameters $\{\theta_{lb1},\ \theta_{ub1},\ \theta_{lb2},\ r_{\max}\}$ were optimally tuned through experimentation in Gazebo.

1.3.2 Approaching the Dynamic Blocks

Once the decision strategy (Subsection 1.3.1) was selected, the next task of the robot is to approach the block from any arbitrary starting position in a manner from which the block can be accurately gripped. We observed through testing that if we try to approach a block from a starting configuration that is too low, the robot arm will often collide with blocks in its path fashion. To avoid this issue, each time the decision algorithm decides on picking a specific dynamic block, we force it to go to the configuration $q = [1, 0, 0, 0, 0, 15]$, which puts the end-effector right above the dynamic table.

Next, we repeatedly check if the selected cube has a pose which allows the end-effector to point downwards with the gripper axes aligned with the block orientation. If the cube is in such a pose, then the robot approaches it through two way-points: first, the robot is placed directly above the block with its grippers matching the orientation of the block. This forces the end-effector to reach the correct pose before coming into contact with the block. Once the L2-norm of the difference between the current and desired configuration is small, the robot moves to the next waypoint, where the gripper drops down towards the block.

The greatest challenge comes from the fact that the waypoints mentioned above are continuously changing over time, since the blocks on the dynamic table are moving. We accounted for this by continuously applying the inverse kinematics function to see if the robot can reach any position that allows the robot to pick up the block. If the robot cannot reach any of the desired poses to pick up a block, the algorithm exits the dynamic block picking condition.

When applying the previously defined strategy, we noticed that the robot lagged behind the block and (with the exception of the dynamic block positioned near the center of the table) was never able to reach the final desired poses. We theorized that this is a result of the lag from constant calculation of the inverse kinematics of the desired pose throughout the approach process. To address this issue, instead of targeting the desired gripping pose at the current state of the dynamic blocks, we targeted the pose of the end-effector corresponding to the pose that the block will be in in a specific amount of time. We introduced a turntable hyperparameter t_{future} which specifies this time. As a function of this time we predict the desired pose of

the end-effector of the robot as follows:

$$T_{\text{future}} = \begin{bmatrix} R_{\text{future}} & p_{\text{future}} \\ 0 & 1 \end{bmatrix},$$

where

$$R_{\text{future}} = \begin{bmatrix} \cos(\Delta\theta) & -\sin(\Delta\theta) & 0 \\ \sin(\Delta\theta) & \cos(\Delta\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot R_{\text{current}} \text{ and } p_{\text{future}} = \begin{bmatrix} x_{\text{future}} \\ y_{\text{future}} \\ z_{\text{current}} \end{bmatrix}.$$

Here, ω_z is the angular velocity of the block, $\Delta\theta = \omega_z \cdot t_{\text{future}}$, $x_{\text{future}} = r\cos(\theta_{\text{current}} + \Delta\theta)$ and $y_{\text{future}} = r\sin(\theta_{\text{current}} + \Delta\theta)$. θ_{current} and r describe the polar coordinates of the center of the block relative to the world frame, and the current values are values extracted from T_{current}, the current pose of the block.

1.3.3 Gripping the Dynamic Blocks

Once the end-effector of the robot is in position for gripping (Subsection 1.3.2), the robot must grip the dynamic block and quickly move away from the dynamic table to avoid collision with the table, other blocks, and the opponent robot. To grip the block, we set the final joint to a value of -5. We then command the robot to raise the end-effector vertically by 30 mm to quickly exit the dynamic table location. We then check if gripping was successful by checking whether or not the targeted block has reached a certain height above the table. If successful, the robot moves to stack the block (Subsection 1.4), otherwise the robot re-calculates the next block to pick (Subsection 1.1.2).

1.4 Stacking

Our strategy for stacking was to place the block as close to the center of the goal platform as possible and to create only a single stack. The first step for stacking is isolating all blocks that are already on the goal platform using their $[x, y, z]$ coordinates and calculating the height of the stack as the z-position of the highest block + 10 mm. If there are no blocks on the goal platform, then the center of the stack is defined as the center of the goal platform. Otherwise, the center of the stack is defined as the center of the block at the bottom of the stack. The primary orientation for stacking is one where the y-axis of the end-effector points in the positive z-direction in the robot frame and the z-axis of the end-effector is pointing towards the center of the stack. The orientation of the x-axis is then calculated as the cross product between these two axes. With this end-effector orientation, we first command the gripper to be located 5 mm above the stack, then descend to 0.5 mm above the stack using methods outlined in Section 1.1.2 before opening the grippers. The robot then safely exits the stack area by subtracting 0.4 radians from the joint 2 state without changing any other joint, ensuring that the robot does not knock over the stack during movement.

Test	Time (s)	Score	Stack Height	Stat. Blocks	Dynam. Blocks	Failed Picks
1	28.33 ± 2.35	19 ± 4.32	73.33 ± 9.42	3.67 ± 0.47	N/A	0.33 ± 0.47
2	37.67 ± 2.05	82 ± 13.49	93.33 ± 9.42	N/A	4.67 ± 0.47	0.66 ± 0.47
3	80 ± 7.11	144.33 ± 6.66	166.67 ± 11.55	3.33 ± 0.58	5.00 ± 0.00	1.00 ± 1.00
4	60 ± 0	61 ± 21.52	106.67 ± 41.63	3.00 ± 1.73	3.33 ± 0.58	0.67 ± 0.58

Table 1: Summary of average simulation time to complete task, total score, stack height, number of static blocks on goal platform, number of dynamic blocks on goal platform, and number of unsuccessful picks (where the gripper attempts and fails to pick a block) over 3 trials for each test condition, with block randomization between each trial. Test 1 evaluates our robot when only allowed to interact with static blocks, Test 2 evaluates our robot when only allowed to interact with dynamic blocks, Test 3 evaluates our robot without a time constraint or opponent, and Test 4 evaluates our robot in a 60-second timed scenario with an opponent robot.

2 Evaluation

2.1 Test 1: Picking and Stacking Static Blocks

We tested our robot without the 60s time constraint and forced the decision function to pick static blocks only. In each of the 3 trials, the robot was able to consistently pick up most of the blocks in less than 30 seconds. The only condition in which picking up all static blocks is not guaranteed is when a block lies outside the reachable workspace of the wrist, where our robot would need to pick it up from the side. In those conditions, the robot may push the block off of the static platform while approaching from the side. These conditions are also the sources of unsuccessful static block picks. The side bonus is largely dependent on the block randomization, since the robot does not intentionally rearrange any of the blocks.

2.2 Test 2: Picking and Stacking Dynamic Blocks

We tested our robot without the 60s time constraint and forced the decision function to pick dynamic blocks only. In almost all trials, the robot was able to pick and stack all 5 dynamic blocks in less than 40 seconds; in one of the trials, the gripper picked up a dynamic block on the diagonal, resulting in insufficient contact between the gripper fingers and the block, and the block was dropped before reaching the goal platform. Again, the side bonus is largely dependent on the block randomization.

2.3 Test 3: Untimed Trials without Opponent

We tested our robot without the 60s time constraint and evaluated our robot's performance. In all three trials, the robot picked up and stacked all 5 dynamic blocks and at least 3 static blocks. The inability to pick up all 4 static blocks again stemmed from conditions where a static block's position is out of the robot

wrist's reachable workspace. The stacking function worked well - in all trials, the robot successfully stacked all blocks it picked up without knocking over the stack. However, the time it took for the robot to stack all reachable blocks was an average of 80s, and thus it is infeasible for the robot to pick and score all 9 reachable blocks within the 60 second timed window.

2.4 Test 4: Timed Trials with Opponent (Competition Results)

During the competition, we played timed matches against opponents in three separate trials, and bested our opponents score-wise in each trial. Even when opponents tried the dynamic block sweeping strategy, our robot's carefully tuned parameters allowed our robot to be quicker in picking dynamic blocks, which proved to be key in ensuring our team's victory during competition; in each trial, our robot was able to consistently pick up at least 3 dynamic blocks. One key difference between this evaluation and the others is the presence of the opponent robot - on several occasions, our robot grippers collided with our opponents', which caused several seconds of confusion in our controller function before our robot was able to continue picking blocks. Interactions with the opponent robot also caused picking dynamic blocks to be less consistent, resulting in imperfect stacking. In one such case, where our robot was able to pick all 9 reachable blocks, the imperfect stacking caused the 9-block stack to topple, resulting in a lower score for that trial.

3 Analysis

Overall our controller is quite versatile, aided greatly by its quickness in moving between picking and stacking configurations. The main controller decision function worked very well, ensuring that the robot picked up feasible dynamic blocks quickly but switched to static blocks when more efficient. In competition and during our untimed trials (Table 1), this allowed our robot to pick up most of the available blocks and place them on the goal platform. The error-handling in the controller to check for successes at each sub-task allowed for the robot to move efficiently as well, ensuring that no time is wasted if an intermediate sub-task was not completed. Consistency and precision in stacking was key for our robot's success in competition - this allowed our robot generally to stack all picked blocks in a single stack on the goal platform, which increased our scores dramatically. We built a solid controller that we are very proud of. Identified limitations in our controller and future work to address these limitations are described below, and alternate strategies we considered but did not implement are described in the Appendix.

3.1 Limitations and Future Work

3.1.1 Imperfect Stacking Position

Our stacking method was deterministic and not dynamic, so small imperfections in each block's position on the stack propagate upwards on the next block. Any imperfection in gripping location would cause slight

errors in the stack since the stacking method assumes the block is fully within the gripper. This is especially problematic because our controller aims to place all picked blocks in one stack, and thus imperfect stacks lead to greater risks of toppling the stack. In future iterations, we would instead use a dynamic stack center based on the center of the highest block to avoid cascades in imperfection that may result in the stack tipping and a major loss in points.

In the future we would also like to work on an algorithm to determine when to start a second stack. We would base this on the mean and standard deviation of the center of the current stack, where when the standard deviation of the centers of each cube in the stack is greater than some parameter, a second stack is created. We would also want to experiment with different re-stacking techniques; this would involve picking the dynamic blocks and placing them at an intermediate position, before re-stacking them.

3.1.2 Orientation of Stacked Blocks

One major limitation of our current controller is the lack of control in orienting all blocks with the white side facing up for the side bonus. Our side bonuses were largely dependent on the randomization of blocks, where we would obtain the side bonus if the static blocks were oriented such that the gripper could pick them up with the end-effector's +y-axis facing the white side of the block.

In the future, we could add additional functionality to our controller emulating Team 8 to ensure we get the orientation bonus on each block. This team used the wrist and static platform and a number of 90 degree rotations of the blocks to re-orient the block with the white side up before placing the block on the stack. We would also need to increase the speed of our controller, likely through using the lynx.command() function, to allow ample time for picking, re-orienting, and stacking within the 60s time constraint. To ensure we secure as many dynamic blocks as possible, we would likely first pick all dynamic blocks before re-orienting all blocks. This would also give us the opportunity to place the static blocks on the goal platform first, after orienting them, and then placing the dynamic blocks on top, so we can get the height multiplier on the more valuable dynamic blocks, thereby increasing the impact of the height of our stack on our score.

3.1.3 Imperfect Dynamic Block Picking

Our method for picking dynamic blocks relies on predicting the future position and orientation of the block based on its angular velocity. The method relies on a time parameter which is used to derive the future position and orientation of the moving block. While we tuned this parameter experimentally to achieve the best performance, the arm does not pick blocks perfectly due to slightly incorrect predictions of orientation or poor grasping. This may lead to further downstream issues in stacking imperfectly picked blocks.

In future iterations, we would like to develop a method to predict the time parameter, based on the code run-time time of the latest block state call; we would develop a method to predict the position of the block at

the current time step based on its velocity and position at a previous time step and the time elapsed between these time steps.

3.1.4 Unreachable Static Blocks

Our method for picking static blocks that are outside of the feasible picking domain of the robot wrist is sub-optimal and often fails due to the need to pick these blocks from the side. We observed during testing that when the robot aims to pick up such blocks from the side, it approaches the block from the top, causing scooping motions rather than gripping motions, which often pushes the block off the static platform. In the future, we could develop an algorithm that defines several waypoints for the robot to move through before gripping the block to ensure it approaches from the side first. The first waypoint would likely align the z position of the gripper with the center of the block, and the second waypoint would then be to move towards the center of the block from this low position. By moving slowly between these two waypoint, we would ensure that the robot approaches the block from a feasibly low position and does not knock the block off the static platform.

3.1.5 Collision with Opponent Robots

During the competition, we observed that colliding with the opponent robot leads to a few seconds of lag with the internal controller of the robot. For example, after colliding with the opponent during Round Robin competitions, our robot performed maneuvers that were not programmed in our controller, going to seemingly random configurations for several seconds before returning to picking and stacking blocks. In the development of our controller we did not consider collision with the opponent robot at all - given that the only overlap between the two robots' reachable workspaces is within a small domain near the dynamic platform, we attempted to avoid collisions by entering and exiting the dynamic platform domain quickly. However, our performance during competitions indicated to us that further collision detection may be beneficial to ensure consistent and efficient performance of the robot. In the future, we would develop a strategy that would allow our robot to actively and dynamically avoid the other robot, perhaps by using potential field planning to plan around the opponent robot.

4 Appendix: Alternative Methods for Dynamic Decision Strategies

Predicting time intervals in which gripping blocks are feasible

In developing our dynamic block decision strategy, we considered using prediction of two time values: the time that a given block will spend in a feasible picking domain and the time it takes for a block to enter the feasible picking domain. We had multiple approaches for predicting the times for each block and defining feasible picking domains.

In the first approach, for each block we interpolated waypoints representing the block's pose around a revolution on the dynamic platform. For each of these waypoints, we used inverse kinematics to determine if the pose is feasible. Once each waypoint for each block is labeled, we could deduce the feasible picking domains of each block through the patterns of the way-points labelled feasible and infeasible. This method is very computationally expensive, as it requires re-calculating feasibility domains for each block at each decision stage.

In the second and third approaches, we wanted to limit the computational requirements. Instead of defining the feasibility domains for individual blocks, we defined global feasibility domains as function of θ and r. We calculated the upper limit of the domain in which the end-effector of the robot could reach a cube with its axis aligned with the negative z axis. Naturally, the lower limit of the domain is that of the table and so we defined the feasible picking domain as follows:

$$\mathcal{F}_{\mathcal{D}} : \sqrt{x^2 + y^2} < 100 \wedge \sqrt{(x-200)^2 + (y-200)^2 + z_{\text{block}}^2} < 331.25,$$

where the former inequality defines the bound set by the dimensions of the table and the latter inequality defines the limits set by the dexterity of the robot, and z_{block} is the height of the top of the blocks on the dynamic platform with respect to the base frame. Note that the bound of the latter inequality was calculated as follows: $L_2 \cdot \sin(1.4) + L_3 = 331.25$, where L_2 and L_3 are the applicable link lengths and 1.4 is the upper joint limit of θ_2.

The biggest issue with how $\mathcal{F}_{\mathcal{D}}$ is defined is that while the former inequality can easily be defined in our desired parametric form ($\sqrt{x^2 + y^2} = r$), the latter inequality cannot be. Our first approach to tackle this hurdle was to approximate the lower limit of the of the inequality using an equation of the following form:

$$r(\theta) = a + b\theta + c\theta^2 + d\theta^3,$$

where the coefficients were left to be defined. The approach we was to simply apply a gradient descent algorithm to minimize the following loss function:

$$\mathcal{L} = \left(\sqrt{(r(\theta) \cdot \cos(\theta) - 200)^2 + (r(\theta) \cdot \sin(\theta) - 200)^2 + z_{\text{block}}^2} - 331.25 \right)^2$$

across a range of points. We defined these through interpolating between the upper and lower limits of θ; the limits of θ are defined through the intersection between the limits of the two inequalities in $\mathcal{F}_{\mathcal{D}}$. The downside of this approach is that to calculate the time in and until this approximated feasible picking domain, we would have to solve for θ from $r(\theta)$.

Our final approach was to combine the aforementioned approach with the initial approach. As before, we interpolate the way-points of each block. However, we now assign only the positions of these blocks to these way-points (as opposed to assigning their poses as well), and simply check whether or not these way-points

are elements of $\mathcal{F}_{\mathcal{D}}$ and proceed accordingly. The benefit of this method is that it is easier to debug and it is computationally far more feasible than the first approach. We thus theorized that this approach would be feasible to apply at every single decision stage.

Once the feasibility domains were defined we applied the following equation to predict the desired times:

$$t_{\text{in feasibility}} = \frac{\theta_{\text{ub}} - \theta(0)}{\omega_z},$$

$$t_{\text{till feasibility}} = 0, \text{ if } \theta(0) > \theta_{\text{ub}}$$

and

$$t_{\text{till feasibility}} = \frac{\theta_{\text{lb}} - \theta(0)}{\omega_z}, \text{ if } \theta(0) < \theta_{\text{ub}},$$

where θ_{ub} and θ_{lb} are the (location) angles of the upper and lower bounds respectively, as defined earlier. $\theta(0)$ is the initial (location) angle of a block and $t_{\text{in feasibility}}$ and $t_{\text{till feasibility}}$ are the times that a block will spend in the feasible picking domain and that the block needs to reach the feasible picking domain respectively, in the current cycle of the dynamic table.

Limitations of these methods

The first approach is limited by its computational complexity and was infeasible for a real-time planner. We spent some effort on optimizing for speed, but realized we needed a simpler strategy. Our second and third approaches calculated feasibility domains based on the reachable workspace of the wrist center. The first approach is sub-optimal in that in each decision stage we would have to solve for θ from $r(\theta)$ and we did not immediately find a function that does this for us automatically. While the second method ran in real-time, we noticed that the arm usually failed on the edges of the feasibility domain. We reasoned that this was due to low maneuverability near joint limits. Ultimately, we decided not to use these strategy of predicting the required times as we realized that making the decision process time-based is far less intuitive than our position-based approach (Section 1.3.1).