MEAM 5200 - Final Project - Franka Emika Pick and Place

Shreyas Raorane Kabir Ram Puri Dhyey Shah Chris Spletzer

1 Scope

Many Roboticists work on the challenge of having robots pick and place objects. One of the most fundamental things that us humans do so efficiently, we attempt to replicate this with robots. This action has the most widespread applications in all industries from manufacturing to farming. As we learn more about the Kinematics involved in making this possible, this challenge gave us a great opportunity to try to approach. Picking Stationary and Non-stationary blocks, and placing them in a stack with a goal of achieving the highest stack possible within a time constraint. The project was an attempt to find the best approach to maximize points, which is also directly correlated to a system which can better address real-world problems. To find the best strategy, in this project, we attempted with various techniques, and thus the scope of this project is appropriate to the challenge in hand.
This report is a discussion of our methodology which includes an overview of our finalized strategy and other methods of interest. Following this, we look into an evaluation of our methods through various tests to determine the performance of these algorithms. An analysis of the results tells us about the efficacy and reliability of the methods which were tested, and lastly we talk about what we learned and the reasons behind our final strategy.

2 Methodology

2.1 High Level Architecture

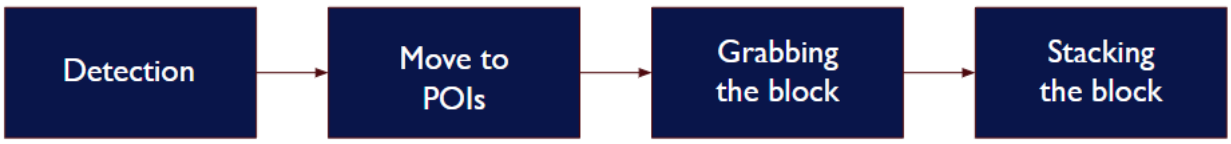

Figure 1: High Level Architechture

Our high-level architecture for the robot control followed a structured flow to efficiently stack static and dynamic blocks into a single tower. For both the static blocks as well as the dynamic blocks, the robot

transitioned from a stationary position to the viewing position, using inverse kinematics (IK) to compute precise joint angles to reach each position. The gripper was controlled to grab blocks with a consistent force of 52N. The gripper was opened at the viewing position before followed by detecting the blocks before it moved to the picking position. Stacking was achieved by incrementing the z-coordinate for subsequent layers.

For dynamic blocks, the strategy incorporated real-time detection and tracking, ensuring the robot adapted its movements to the block's updated position. By stacking four static blocks first, followed by four dynamic blocks, the robot maintained stability and precision throughout the task. One of our main goals with this sequence was to reduce computational and stationary times while minimizing uncertainty. The computational costs were minimized by storing configurations and paths of interest which the robot often traveled to prevent repeated calculations of IK solutions. Additionally, an optimal number of locations of interest were found that prevented the robot from stopping unnecessarily while making sure that all required detections could be taken.

Algorithm 1 Pick and Place algorithm

```
1:  procedure Main
2:      Initialize robot and set start position
3:      Define pickup and drop positions
4:      PickPlaceStatic(pickup_position, drop_position, 4)
5:      PickPlaceDynamic(rotate_position, drop_position, 6)
6:  end procedure
7:  procedure PickPlaceStatic(pickup_position, drop_position, blocks_to_stack)
8:      while blocks remain AND stack incomplete do
9:          Detect and locate static block
10:         Pick up block
11:         Move to drop position and place block
12:     end while
13: end procedure
14: procedure PickPlaceDynamic(rotate_position, drop_position, blocks_to_stack)
15:     while True do
16:         Detect and track rotating block
17:         Pick up block from rotating platform
18:         Move to drop position and place block
19:     end while
20: end procedure
```

A detailed version of the above algorithm can be found at the end of the report in the appendix.

2.2 Direct IK method

To move the end effector from one pose to another, we tested with the Direct IK method and the RRT method. In this section, we discusses the Inverse Kinematics solution being used to directly to determine the path of the robot arm.

When we are interested in finding combinations of joint angles that achieve a certain end effector location,

we perform gradient descent from a seed configuration and converge to our desired configuration. within this, we can introduce certain secondary tasks, in this case, which was keeping the joints as close to the center of the joint limits as possible.

At every step of gradient descent, we can document the configuration and directly use this path in the configuration space to move from one position to another. Sometimes the IK fails to find a valid solution due to a number of reasons, this can cause the arm to assume an unusual orientation that can be in the path of a collision. To prevent this, certain checks were added that detected such failures and prevented any movement and forced either recalculation or skipping that block which might cause such a problem. This was the method that was implemented in our final pick-and-place challenge.

2.3 Rapidly Exploring Random Trees (RRT)

In this section, we discuss the second method that we explored to pick and stack blocks i.e. using a bidirectional RRT path planning algorithm. The primary objective for this method comprised of planning collision-free trajectories in the robot's configuration space between a start configuration q_block(the position of a block to be picked) to a goal configuration q_place(the desired placement position of the block) using RRT. We also removed the intermediate position q_drop as compared to out IK methodology for this approach which was intended to optimize the RRT path and leverage its capabilities of path planning without an extra set point to avoid direct collisions with the stack. Additionally we made sure that the robot operates within the constraints of joint limits and must avoid self-collisions throughout the implementation.

Our robot states are represented as joint configurations where each joint angle lies within predefined limits [q_min, q_max]. The function is_self_collision(q, fk_instance) validates if a configuration q results in a self-collision or violates joint limits using forward kinematics to compute joint positions and ensures no two joints are closer than a predefined tolerance. Random configurations are sampled uniformly within joint limits with a goal_bias to encourage sampling towards the goal, which thereby improves our convergence efficiency. For each sampled configuration, the nearest neighbor in the tree is determined, and a new configuration is generated using a steering function constrained by a defined step size much similarly to our implemented RRT algorithm logic learnt from Lab 4. When the trees connect, the solution path is extracted. The resulting path is executed by moving the robotic arm sequentially through the configurations using the arm.safe_move_to_position(config) command.

2.4 Block detections and Complementary filter

In this section, we discuss the approach used for detecting the pose (Rotation and Translation) of the blocks, both Static and Dynamic, each marked with AprilTags and transforming their coordinates into the world and end-effector frames as needed using a series of transformations and a complementary filter function to improve the robustness of our algorithm for hardware implementation.

We begin with obtaining the homogeneous transformation matrix H_ee_camera, which represents the camera frame in terms of the end-effector frame (H_c_ee) by the detector.get_H_ee_camera() function. The end-effector's position in the world frame is obtained through forward kinematics, represented by H_ee_w. By multiplying H_ee_w with H_c_ee, we then calculate the camera pose in the world frame (H_c_w). Finally we iteratively retrieve the block pose in terms of the camera using detector.get_detections() before being passed to a complimentary filter (explained below) where the consecutive readings are stored in b_reading_1 and

b_reading_2 after which we multiply H_c_w with the filtered b_reading_comp reading to get the pose of the blocks in term of the world frame returned as block_pose_world. This array contains the stabilized and transformed positions of all detected blocks in the world frame.

Figure 2: Block detection in Simulation and Hardware Testing

To improve the stability and accuracy of block detection from the detector.get_detections, we used a complementary filter, where the function comp_filter() function smooths the consecutive detected block positions by blending the current reading with the previous reading using a weight factor α. (According to the equation filt_pose = (filtered_pose=α*current_reading+(1−α)*previous_reading)). This iterative filtering process reduces noise and stabilizes the block coordinates over multiple detections, improving the overall reliability of the detected pose.

2.5 Orienting the End-Effector

The above pipleine enabled us to retrieve a correct and reliable pose for the blocks and the necessary transformation for the end-effector to correctly pick it. The get_rotation_z_angle(rotation_matrix) function processes a 4x4 homogeneous transformation matrix to compute the rotation angle about the z-axis (rz) that the joint 7 (q[6]) has to rotate for orienting the end-effector to pick the block perfectly.

The 3x3 rotation matrix is extracted from the top-left corner (Rotational matrix) of the input matrix obtained from our block_pose_world, and the absolute values of its elements are calculated. The get_rotation_z_angle function identifies the column closest to [0,0,1] (indicating that these axis are aligned in the same direction or exactly 180 degrees opposite for the block and end-effector) and swaps it with the z-column (third column) of the rotation matrix. After reordering the matrix columns using swap_columns, the corrected rotation matrix is used to compute the z-axis rotation angle using the atan2 function. This calculation utilizes elements of the first column to determine the angle about the z-axis in radians. To maintain a consistent range for the angle, it is adjusted to lie within $[-\pi/2,\pi/2]$. The resulting angle is returned for further use in orienting q[6], providing an accurate and consistent measure of each of the block's orientation.

2.6 Pick-Place Algorithm

Static Blocks

For picking and placing the static blocks, our methodology was simple but high repetitive as the static blocks would make the base for our tower. The PickPlaceStatic(pickup_position, drop_position, blocks_to_stack)

was used to first move the end-effector to the q_above_pickup (pickup_position) which was the perfect position for all the static blocks to be in the frame for the camera detection. Next, the block detection function was executed to detect the static block poses which it returned in block_pose_world. Next, the gripper was opened with a consistent force of 52N before the end-effector moved to grab the block with the correct orientation (q[6]). After a static block was grabbed, the end-effector was moved to q_drop (drop_position). This process was repeated 4 times consecutively for the 4 static blocks (blocks_to_stack = 4), and the z height was incremented iteratively by 0.05m to stack the blocks above each other. Once this loop runs for 4 times, the end-effector proceeds over to the viewing position on top of the turntable.

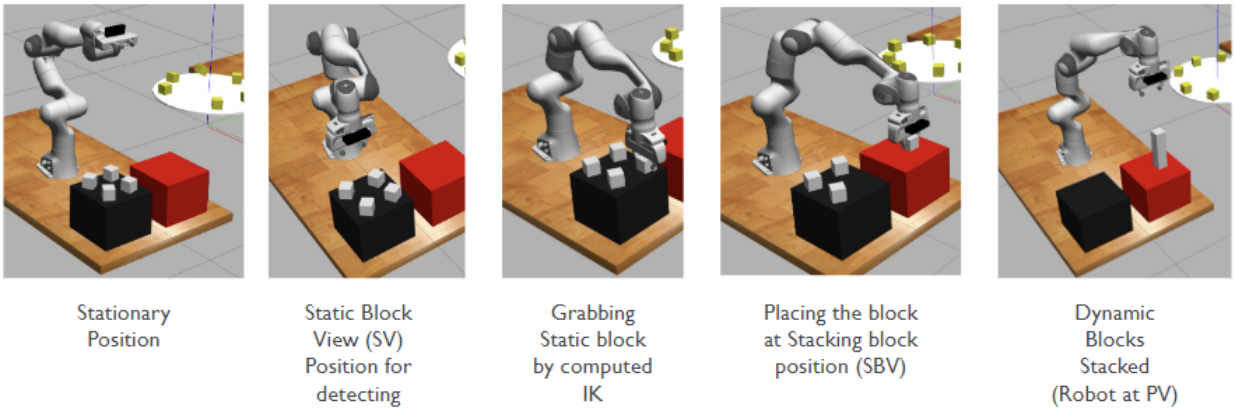

Figure 3: Pick and Place Static Blocks in Simulation

Dynamic Blocks

Picking and placing the dynamic blocks required comparatively more precisely timed detection, prediction, and motion execution of the robot arm. Similar to the static blocks strategy, for the dynamic blocks, the PickPlaceDynamic() method was used. We being by moving the end-effector above q_above_rotate (rotate_position) to detect the dynamic blocks. After the camera detected a block, its Cartesian coordinates (x,y) relative to the turntable center, these coordinates were converted to polar form (r,θ). Given the turntable's constant angular velocity (ω) and known radius of r=0.305m, the block's motion was assumed to follow uniform circular motion. To predict the block's future angular position, the equation $\theta(t)=\theta(0)+\omega t$ was used, where ωt was tuned based on observed angular velocity and response time. This predictive model allowed the robot to calculate the optimal time and position to intercept the block. By precisely timing the motion of the end-effector to the block's predicted location and synchronizing it with the gripper's action, the robot successfully grabbed the dynamic block in motion. For the competition, ωt was set 0.24 which we tuned by hit and trial to ensure correct timing of the picking. Once secured, the block was moved to q_above_drop_stacked (stacking position) and then placed above the stack with the fixed z height(incremented with each iteration) accurately.This process was repeated 4 times integrating the dynamic block seamlessly on top the tower structure(4 existing static blocks).

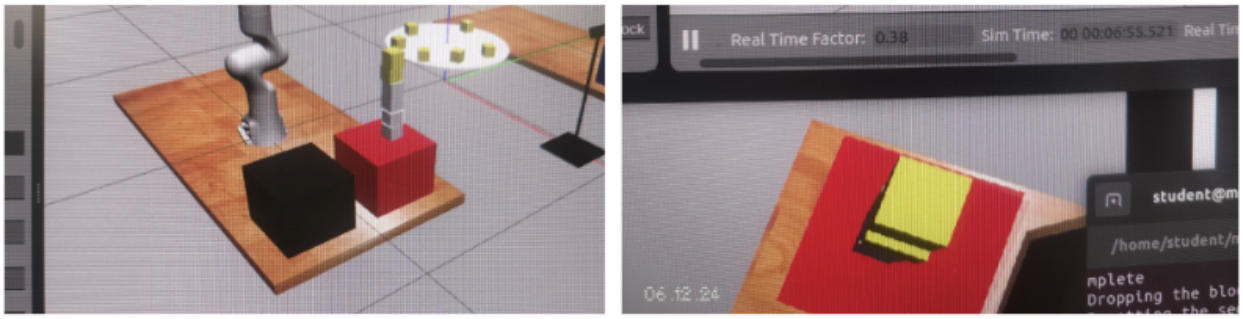

Figure 4: Stack with dynamic blocks in simulation

2.7 Soundness and Validity of Methodology

The strategy was to place the static blocks reliably. We had a very high success rate in picking up the static blocks and placing them exactly where we needed them to go. This gave us confidence that the base of our structure would be secured which would allow us to stack Dynamic blocks on top of them to maximize the points.

Our approach was technically sound, leveraging established kinematic principles and optimization techniques and pre-storing configurations to increase computational efficiency while maximizing precision. The system's design was reproducible due to clear algorithm definitions, consistent procedures, and well-documented parameters.

The approach covered all essential tasks, including block detection, path planning, and error handling. Implementing fallback mechanisms for IK failures enhanced system reliability. Potential biases were mitigated by using dynamic recalculations and testing across diverse scenarios. However, occasional limitations arose from environmental factors like sensor noise, highlighting areas for future improvement.

By balancing efficiency, robustness, and adaptability, the methodology provided a comprehensive solution for robotic pick-and-place tasks while ensuring scalability and real-world applicability.

3 Evaluation

3.1 Testing Methodology

The testing process was conducted through extensive evaluation in both simulation and hardware environments. The simulation environment provided ideal conditions for initial testing and validation, allowing us to achieve near-perfect results, particularly with static block manipulation. Testing included factors such as end-effector positioning accuracy, block stacking precision, time efficiency for both static and dynamic blocks, and system reliability.

Hardware testing focused on real-world performance factors and system robustness. The methodology in-

corporated continuous refinement of control parameters and emergency 'p'-values whereever possible to ensure safe and efficient operation. These emergency values were predefined values the Robot would take if at all the Inverse Kinematics solution was not able to converge.

Testing metrics included pick-and-place accuracy, stack stability, time efficiency, and error recovery mechanisms.The computational costs were minimized by storing configurations and paths of interest which the robot frequently traveled, preventing repeated calculations of IK solutions.

Videos for our initial testing on simulation and hardware can be found below:
Simnulation Testing Video: Simulation Testing Stack
Hardware Lab Testing Video: Lab Testing Stack

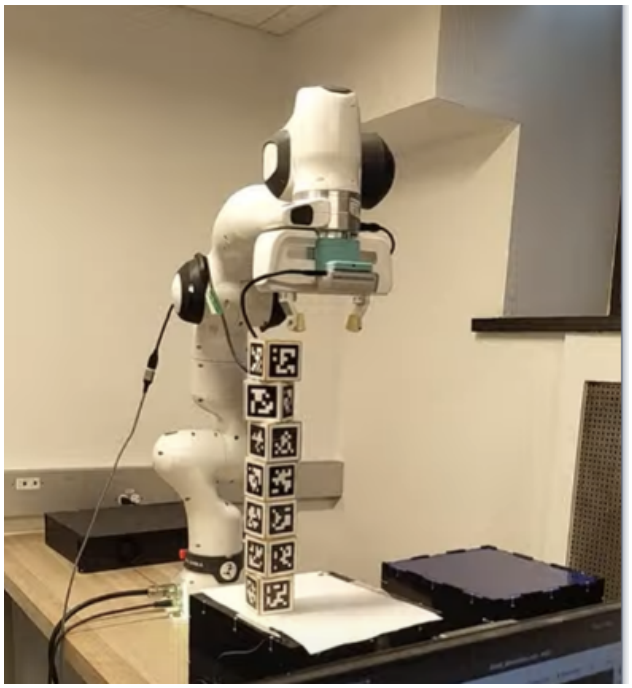

Figure 5: Ideal Stack from Hardware testing in Lab

3.2 Results

3.2.1 IK Results:

Run	Static Stack Time (s)	Dynamic Stack Time (s)	Total Run Time (s)
1	117.974	131.443	256.67
2	112.981	131.894	251.27
3	113.953	139.042	258.38
4	110.522	140.634	259.02

Table 1: Stacking Times for Static (4 blocks), Dynamic (4 blocks), and Total Runs (8 blocks)

3.2.2 RRT Results:

Run	Static Stack Time (s)	Dynamic Stack Time (s)	Total Run Time (s)
1	154.491	246.573	408.26
2	154.521	250.893	412.27
3	157.065	242.496	408.22
4	156.206	247.409	410.36

Table 2: Stacking Times for Static (4 blocks), Dynamic (4 blocks), and Total Runs (8 blocks)

3.3 Overall Evaluation

We explored 2 major approaches to solve the task at hand:
1. Pure IK Approach
2. RRT Approach
As the Results indicate, the IK Approach turned out to be much faster when compared to the RRT. The ask was to complete the task within 5 minutes (300 seconds), for which the only viable option was to go with the IK Approach.

Several failsafes were added to the code which made sure that if the IK failed, we just move on to the next block and attempt to get the IK for that. The code is robust in the terms that several checks have been added for the safety of the Robt as well as the operators.

The video of the stack made during the competition can be found here: Competition Stack

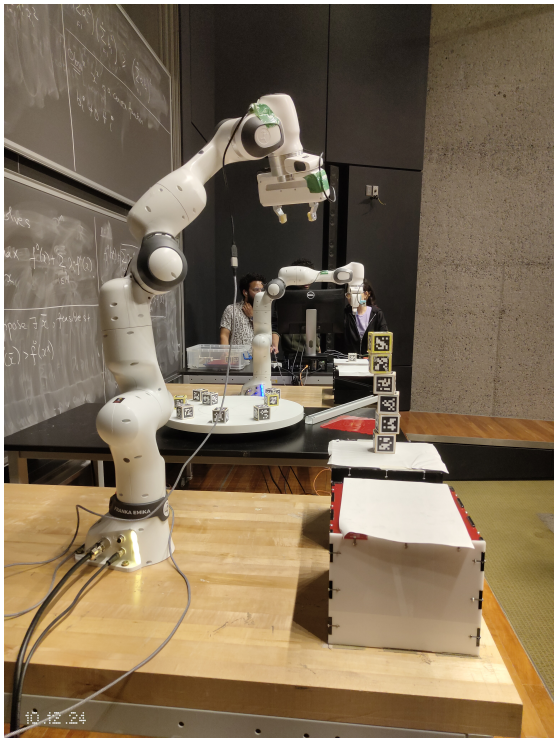

Figure 6: Stacking Process

4 Analysis

4.1 IK vs RRT

The Direct IK method demonstrated superior performance compared to RRT in both simulation and hardware testing.

Simulation:

- Static Block Success Rate: 100%
- Dynamic Block Success Rate: 85%
- Average Total Run Time: 256 seconds

Hardware

- Static Block Success Rate: 80%
- Dynamic Block Success Rate: 60%
- Average Total Run Time: 410 seconds

Time Efficiency

Operation Type	IK Method (s)	RRT Method (s)
Static Stack	115s	155s
Dynamic Stack	135s	246s

Table 3: Time Efficiency

The IK approach proved particularly effective for establishing a strong foundation with static blocks, though dynamic block placement showed slightly reduced accuracy due to timing and mechanical constraints.

RRT, while theoretically sound, exhibited longer execution times and lower success rates. In simulation testing, RRT achieved an 80% success rate with static blocks but only 60% with dynamic blocks. The extended computation time for path planning made RRT impractical for meeting the 5-minute competition constraint, as evidenced by total run times exceeding 400 seconds compared to IK's approximately 260 seconds.

The IK method proved more practical for competition requirements, as it proved by successfully stacking 5 blocks (4 static, 1 dynamic) within the 5-minute time constraint. The RRT method was not implemented in hardware due to its prohibitive computation times.

4.2 Overall Analysis

The competition performance validated our decision to implement the IK-based approach. The system successfully stacked 5 blocks (4 static, 1 dynamic) during the competition, demonstrating reliable performance with both red and blue blocks. Our strategy earned us a highest point score of 8500 in the competition which was pretty good had helped us reach the semi-finals.

The implementation of offset adjustments proved crucial for real-world operation, compensating for environmental variables not present in simulation.

The time efficiency of the IK method was particularly notable, with static block stacking consistently completed in around 120 seconds and dynamic block handling in approximately 135 seconds. This efficiency allowed for multiple attempts within the competition time limit, providing opportunities for error recovery and stack optimization.

5 Lessons Learned

The final project served as a perfect culmination of all the concepts we progressively learned through the labs. Seeing the theory we learned come to life in a real-world task during the competitive environment of the final competition was incredibly rewarding. Throughout the project, we developed a comprehensive understanding of modeling and controlling the 7-DOF Franka Emika Panda Arm, along with programming it to perform tasks efficiently.

Along this way, we learnt and realized a lot of different things that contributed to out success in teh competition and added to our knowledge. Firstly, we observed that inverse kinematics (IK) computations can be time-intensive and may fail for certain target poses. To mitigate this, we identified and pre-calculated fail-safe joint configurations, which proved particularly useful during the competition by preventing unwanted robot behavior or collisions. This approach was especially effective for repetitive tasks, such as the pick-and-place sequence. Path planning using techniques like RRT didn't provide significant benefits in relatively obstacle-free environments. Our tests showed that these planners often resulted in less smooth trajectories and increased runtime, as reflected in our results.

Transitioning from simulation to hardware presented notable challenges, particularly due to inconsistencies in the hardware robot and uncertanities in teh environment. Issues like noise in the AprilTag pose detection led us to explore filtering and averaging techniques. Additionally, minor offsets in the camera's position required real-time offset compensation, which we addressed by fine-tuning the offset values the day before the competition. These adjustments ensured smooth operation on both the red and blue sides of the robot.

While our dynamic block-picking strategy faced occasional issues, particularly on the blue side, we realized that a better viewing and picking configuration of the robot above the turntable combined with force feedback from the end-effector would improve robustness and reliability. Tuning the wt parameter at the day of the competition could also have enhanced the performance of dynamic picking.

Increasing the robot's speed introduced challenges as it exceeded position thresholds for our set poses. Consequently, we maintained the speed limits enforced by the arm.safe_to_move() command. However, slightly increasing the speed might have given us a competitive advantage but could have proven risky on occasions

which might lead the IK to fail and the end-effector might have collided with other blocks stopping our competition. Moreover, incorporating trajectory optimization techniques, such as minimum acceleration or minimum jerk trajectories, could have mitigated issues like collisions when the robot operated at higher speeds.

Overall, our single-tower pick-and-place strategy performed well in both simulation and hardware. **Our team placed #3rd in the competition** and managed to stack the tower consistently and we were extremely happy with our performance. With additional fine-tuning, it could become even more robust. Working on the final project as a team allowed us to identify and leverage each other's strengths while fostering collaboration. This experience emphasized the importance of accounting for all possible scenarios and identifying real-world errors that only become apparent through hardware testing, rather than relying solely on simulations.

Algorithm 2 APPENDIX: Detailed Algorithm

procedure Main
    Initialize robot arm and object detector
    Set start position
    $q_{above\_pickup}, q_{above\_drop} \leftarrow$ SetStaticView(start_position)
    $q_{above\_rotate}, q_{above\_drop\_stacked} \leftarrow$ SetDynamicView(start_position)
    PickPlaceStatic($q_{above\_pickup}, q_{above\_drop\_stacked}, 4$)
    PickPlaceDynamic($q_{above\_rotate}, q_{above\_drop\_stacked}, 6$)
end procedure
procedure PickPlaceStatic($q_{above\_pickup}, q_{drop}, blocks\_to\_stack$)
    Move arm to q_{above_pickup}
    while $block\_count > 0$ AND $stacked < blocks\_to\_stack$ **do**
        $block\_count, block\_world \leftarrow$ GetBlockWorld()
        Open gripper
        $q_{align}, q_{block} \leftarrow$ MoveToStaticBlock($block\_world[0]$)
        Move to q_{align} then q_{block}
        Close gripper
        Move to q_{drop}
        $q_{place} \leftarrow$ MoveToPlace($iteration$)
        Move to q_{place}
        Open gripper
        $iteration \leftarrow iteration + 1$
    end while
end procedure
procedure PickPlaceDynamic($q_{above\_rotate}, q_{drop}, blocks\_to\_stack$)
    Move arm to q_{above_rotate}
    while $iteration < blocks\_to\_stack$ **do**
        $block\_count, block\_world \leftarrow$ GetBlockWorld()
        if $block\_count > 0$ **then**
            $q_{block} \leftarrow$ MoveToDynamicBlock($block\_world[-1]$)
            Move to q_{block}
            Close gripper
            Move to q_{drop}
            $q_{place} \leftarrow$ MoveToPlace($iteration$)
            Move to q_{place}
            Open gripper
            $iteration \leftarrow iteration + 1$
        end if
    end while
end procedure
procedure MoveToDynamicBlock($block$)
    Calculate end effector position
    Apply dynamic adjustment for rotating blocks
    Calculate inverse kinematics
    return joint angles
end procedure
procedure MoveToStaticBlock($block$)
    Calculate approach position above block
    Calculate final position at block
    Calculate inverse kinematics for both positions
    return approach and final joint angles
end procedure