ROB 550 Armlab Report

Jackson Chen, Jake Kanetis, Ziqing Qin, Basavasagar Patil
{jshchen, jkanetis, ziqingq, bpatil} @umich.edu

***Abstract*—In response to the escalating demand for autonomous robotic manipulation in various environments, this project introduces a novel integration of computer vision and path planning algorithms to enable an RX200 robotic arm to autonomously manipulate and relocate blocks of different colors. Utilizing a RealSense L515 RGB-D camera, we develop a methodology for accurate block detection, including 3D positioning, orientation, and color identification. The subsequent path planning algorithm employs forward and inverse kinematics to find the efficient trajectory for task execution. Preliminary results demonstrate the system's proficiency in task completion within a constrained setting, highlighting its potential implications for enhancing robotic autonomy and efficiency in similar applications. However, the system's current limitations underscore the necessity for further refinement to achieve universal task completion.**

I. Introduction

At the turn of the century, spending on robotic development and integration was $7.4 billion across multiple industries, with the military and industrial markets leading most of this expenditure [1]. By 2025, worldwide spending on robotics will increase by 10 fold. Studies predict that the industrial market will dominate nearly 25% [1]. Robotics development is on pace to be one of the most prevalent industries in the world. The first industrial robot was installed in 1967, and since then robots, have dominated production lines, particularly in the automotive industry [2]. While production and efficiency has exponentially grown, the concern for human-robot interaction has been at the forefront of robotics development. The term "Industry 4.0" has been coined to describe the Fourth Industrial Revolution, an idea that envisions industrial robots and human working seamlessly side by side in order to respond to the high-volume demands of modern manufacturing needs [3]. These new cooperative environments highlight the importance of designing robots which utilize computer vision and self correcting algorithms to make informed decisions to ensure the safety of their human counterparts.

Computer vision is a staple of the robotics industry and allows robots to make informed decisions with no to limited human guidance. One of the most common applications in manufacturing industry is bin picking and assembly line part identification [4]. In these processes, the robots are performing the act of selecting an item from a bin or off the assembly line and moving it to a new location using stereo-vision or LiDar based detection systems. The robot recognizes the object and generates a path using the defined algorithm. Another use of computer vision is defect detection. This describes the process of recognizing specific features on an item (e.g. a warped thread on a screw) and removing the damaged object from the assembly line [4]. However, limitations still constrain the performance or robotic performance. While defect detection is sought after, image quality issues prevent systems from recognizing variability between similar components. Additionally, lighting changes drastically affect the performance of the systems ability to detect object [4].

In order to explore the environments and challenges industrial robots face, our team designed computer vision and path planning algorithms for a RX200 arm along with a Realsense l515 RGB-D camera for a series of 4 different tasks which replicate common duties an industrial robot would execute. The four tasks involved various combinations of color/size recognition with the forward/inverse kinematics algorithms to implement pick, sort, place, and stack functions. This report discusses the methodology behind the computer vision algorithms, the design of the forward and inverse kinematics of the RX200 arm, and the performance of the procedures we designed for each specific task. Our objective was to emphasize the importance of the computer vision algorithm and design intelligent block detection, as this is directly correlated to challenges faced by industrial robots.

II. Methodology

A. Camera Calibration

The intrinsic matrix is constructed using the offset of the optical center C_x, C_y and the focal length f_x, f_y of the camera (1). It essentially describes a projective transformation from the 3D camera frame to the 2D image frame.

$$K = \begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

The intrinsic matrix can be obtained by two methods. The first method is to use the ROS camera calibration package. A 8×6 checkerboard is moved w.r.t. all its

6 DOFs under the camera until it is sufficient for the camera to calibrate. We repeated this process 4 times and calculated the average intrinsic matrix. The second method is to acquire the factory intrinsic matrix published on the `camera_info` topic by `echo` command.

B. Workspace Reconstruction

To reconstruct the workspace, we need to transform the image coordinates to world coordinates. This transformation requires two steps: the first step is to transform the image (pixel) coordinates (u, v) to camera coordinates (X_c, Y_c, Z_c) with the intrinsic matrix and depth information as shown in (2).

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c(u, v)K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2}$$

The second step is to transform the camera coordinates to world coordinates with the extrinsic matrix H (3). The extrinsic matrix can be obtained with two methods. This process is essentially a rigid-body (or homogeneous) transformation between two 3D frame since there's no stretch or deformation (4).

The first method is to directly measure the translation and rotation between the camera and world frame to construct the extrinsic matrix. Assumptions could be made to simplify the measurement process. However, this method does not guarantee the accuracy of the extrinsic matrix. The second method is to use perspective-n-point (`solvePnP`) function from `OpenCV` to calculate the rotation matrix R and translation vector T given the pairs of world coordinates and camera coordinates. The points we selected for this calculation were the four corners of each of the four AprilTags, with a total of 16 pairs.

$$H = \begin{bmatrix} R & T \\ \underline{0} & 1 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = H^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{4}$$

The work described above illustrates the process of obtaining the world coordinates of the workspace from the RGB camera and depth sensor. At this point, the x and y coordinate values were calibrated; however, the depth values (z coordinates) were skewed relative to the x-y position in the workspace. This occurred because the depth sensor is not aligned with the RGB camera and is tilted with respect to the board. Our team used three x-y points and the points' current z values to fit a plane to the tilt of the depth sensor (5). In this sense, we were able to map the z offset at every point on the board, and thus, were able to subtract this offset from the depth read from the sensor.

$$z_{offset} = ax + by + c \tag{5}$$

Since the camera is not placed at the center of the workspace, in order to get a bird-eye view of the workspace, we need to perform a perspective transformation for our current GUI video frame, which requires a homography matrix (6). Note that h_{11}, h_{12}, h_{21}, and h_{22} are responsible for the rotation and scaling of the transformation while h_{13} and h_{23} are responsible for the translation. Besides that, h_{31} and h_{32} are responsible for the image's perspective and h_{33} are usually set to 1.

$$h = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{6}$$

The homography matrix are obtained using `findHomography` from `OpenCV` by feeding at least four pairs of points to solve for the eight unknown parameters inside the homography matrix.

C. Block Detection

In order to pick and sort the blocks, a block detection algorithm is written to differentiate the blocks by color and size, which returns the orientations and centroid coordinates of the detected blocks.

The block detection algorithm starts with `findContours` for all the possible contours inside the workspace. Since `findContours` can return two or three values depending on the version of OpenCV. Use unpacking to support both cases. Then, to identify contours of the blocks, we first filter out the non-squares by comparing the width and height of the contours using `boundingRect`, then we filter out the noise pixels using a median filter `medianBlur`. After that, all the contours with an area less than 100 pixels are filtered out by `contourArea` to ensure the elimination of noise. Then we find the coordinates of the centroid with `moments` and equation (7). Since it is possible to segment the side of the block as part of the contours, a depth filter is also added to ensure only the top surface of the block is included. The orientation of the blocks with respect to the positive x axis in the world frame is calculated by the `minAreaRect`. And finally, the pixel threshold that differentiates the small and large blocks are determined by trial and error. A unique id is assigned to each block to ensure correct coordinate, area and orientation are stored at the correct location in the dictionary.

$$(x, y) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}) \tag{7}$$

To differentiate blocks with different colors, blocks with the target color are placed in the workspace to determine the HSV threshold of each color. The threshold should be good enough to differentiate blocks with adjacent but different colors, i.e., orange and red. All the Hue, Saturation, and Value threshold should be adjusted to optimize the accuracy under the current lighting condition and are due to change if lighting condition changes.

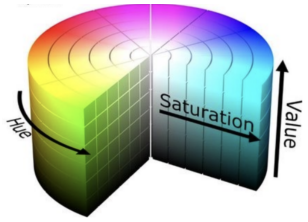

Fig. 1. HSV Values for Different Colors

D. Forward Kinematics

We use Product of Exponentials (PoX) method as our primary method for forward kinematics (FK), although we also implemented DH parameters method for verification. Although the implementation of the DH parameters is efficient, PoX has a number of advantages that led us to use it over DH parameters. PoX method treats uniformly the prismatic and revolute joints and provides an easy geometric interpretation from the use of screw axes for each joint.

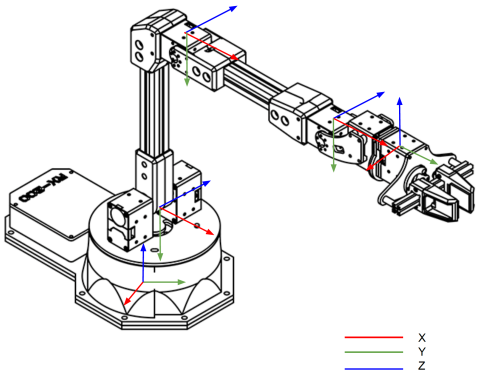

Fig. 2. Screw matrix axes for all the joints for Forward Kinematics

PoX requires a home position matrix, where all the joint angles are set to zero, and a matrix exponential for each joint.

$$M = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 424.15 \\ 0.0 & 1.0 & 0.0 & 303.91 \\ 0 & 0 & 0 & 1.0 \end{bmatrix}$$

$$S = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & -103.91 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & -303.91 & 50.0 \\ -1.0 & 0.0 & 0.0 & 0.0 & -303.91 & 250.0 \\ 0.0 & 1.0 & 0.0 & -303.91 & 0.0 & 0.0 \end{bmatrix}$$

Here M refers to the home position matrix, and each row of the S matrix refers to the screw matrix of a joint. In forward kinematics, we are interested in finding the position of the end effector given the joint angles θ, i.e, we are interested in the function $T : \mathcal{R}^n \rightarrow \mathcal{R}^6$, where n is the number of joint angles.

$$T(\theta) = e^{[\mathcal{S}_1]\theta_1} e^{[\mathcal{S}_2]\theta_2} e^{[\mathcal{S}_3]\theta_3} e^{[\mathcal{S}_4]\theta_4} e^{[\mathcal{S}_5]\theta_5} M$$

In the above equation, $[\mathcal{S}_i]$ for i = 1 to 5, represent a skew symmetric matrix that is obtained by the arrangement of angular (the first three numbers of each row) and linear (the last three numbers of each row) components of the skew matrix for respective joints.

To verify the accuracy of our FK, we calculated the average the error in the x, y, z coordinates at random sample locations given the joint angles readings. The Absolute Error (AE) for the FK was found to be less than **3mm** for each coordinate. The accuracy was achieved by subtracting a fixed offset of 0.033 and 0.024 from the joint angles 1 and 2 to adjust for the fact that our home position values of joint angles were not all zero, most likely due to motor wear down.

E. Inverse Kinematics

Inverse kinematics (IK) describes the process of calculating the joint angles required to achieve a desired end-effector position. During the competition, our computer vision algorithm would identify the position of the blocks in Cartesian space, and our inverse kinematics algorithm would calculate the necessary joint angles to grasp the block. Our team utilized a geometric approach to calculate the joint angles. This approach defines a closed solution and is more computationally efficient than using a numerical method which utilizes an iterative approach to calculate the optimal joint angles. One important feature our team had to consider when designing the algorthim is the fact that the RX200 arm can manipulate its end effector such that it is in the same position but with the elbow joint in the up or down position. Our algorithm only considers the elbow up configuration

as this orientation is more effective at grabbing and transporting block in the given workspace.

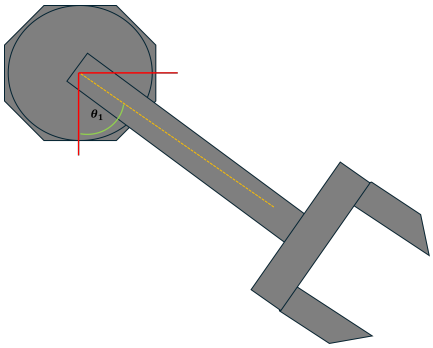

Fig. 3. Top Down View

Figure 3 represents a top down view of the RX200 arm. The first angle that can be calculated is the joint angle of the base. This angle is only dependent on the position of the arm in the x-y plane.

$$\theta_1 = atan2(-x, y) \tag{8}$$

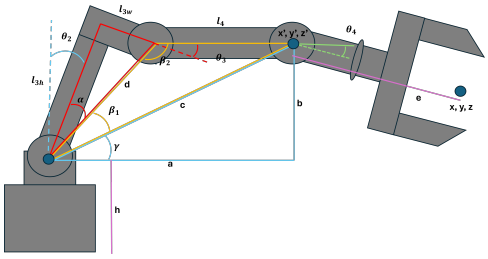

Fig. 4. Top Down View

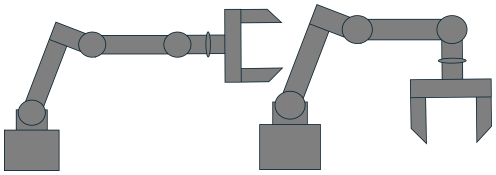

Fig. 5. Horizontal and Orthogonal Grasping Positions

Figure 2 represents a profile view of the RX200 arm. We defined a series of geometric relations to calculate the necessary joint angles. One important consideration was the orientation of the end effector when grasping the blocks. The implemented algorithm only considered a horizontal or orthogonal grasping position (Fig 5). While this approach simplified the calculations, it was also chosen as the blocks are cubic in shape and these two positions are most conducive to confidently manipulating the objects. In our algorithm, we specified this angle as θ_{4i}.

The constant lengths are defined first (all values are in milimeters).

$$d = 205.73, e = 174.15, h = 103.81 \tag{9}$$

$$l_{3h} = 200, l_{3w} = 50 \tag{10}$$

We then translated the end effector's position relative to the θ_4 angle in order to calculate the angles of the previous joints. The geometric relations of the blue triangle are calculated first.

$$z' = z + e * \sin\theta_{4i} \tag{11}$$

$$a = \sqrt{x^2 + y^2} - e * \cos\theta_{4i} \tag{12}$$

$$b = z - h \tag{13}$$

$$c = \sqrt{a^2 + b^2} \tag{14}$$

In order to calculate θ_2, we first must solve α, β_1, and γ.

$$\alpha = \arctan\frac{l_{3w}}{l_{3h}} \tag{15}$$

$$\beta_1 = \arccos\frac{c^2 + d^2 - l_{3h}}{2 * c * d} \tag{16}$$

$$\gamma = atan2(b, a) \tag{17}$$

Thus, θ_2 is the sum of these angles subtracted from an upright postion of the shoulder link.

$$\theta_2 = \frac{\pi}{2} - (\alpha + \beta_1 + \gamma) \tag{18}$$

Solving θ_3 is reliant on our knowledge of the current β_2 and α angles.

$$\beta_2 = \arccos\frac{l_{3h}^2 + d^2 - c^2}{2 * d * l_{3h}} \tag{19}$$

$$\theta_3 = \frac{\pi}{2} + \alpha - \beta_2 \tag{20}$$

The designed algorithm takes the input of the desired orientation of the gripper (θ_{4i}). The value of θ_4 the must be transfer to the FK algorithm and must account for the current θ_2 and θ_3 angles as these orientations will affect the grippers orientation relative to the blocks. Thus, the equation below corrects for any deviations from the orthogonal or perpendicular orientations that occur from the movement of the previous joints.

$$\theta_{4o} = \theta_{4i} - \theta_2 - \theta_3 \tag{21}$$

Seen above are the geometric relationships we utilized to calculate the inverse kinematics. For our path planning algorithm, a desired θ_4 was inputted to represent if the end effector should be parallel or orthogonal to the blocks; this was denoted as θ_{4i} as it was the angle we "inputted". Many aspects of the tasks required the end effector to be aligned with the blocks orientation, thus we sought to manipulate the θ_5 angle to correct for any misalignment. Seen in (22) are the various cases we considered. The angle was set to 0 if we chose to grasp the block with a parallel configuration. If the arm grasped the block from an orthogonal rotation, θ_5 was adjusted in accordance to the current alignment of the base (θ_1) and the measured orientation of the block θ_{5i}. Again, the lowercase "i" represents and input we inserted into the algorithm.

$$\theta_{5o} = \begin{cases} \theta_1 + \theta_{5i} & \text{if } \theta_{5i} \text{ exists and } \theta_{4o} \text{ is not 0} \\ 0 & \text{if statement above is false} \\ 0 & \text{if } \theta_{4o} = \frac{\pi}{4} \end{cases} \tag{22}$$

F. Click and Place

Upon the completion of the FK and IK, we integrated a click and place algorithm. The user first clicks on the top of a block from the GUI, and the mouse coordinates are converted to world coordinates using the inverse of the extrinsic and intrinsic matrices. The z-value was read from the depth sensor and offset by the defined plane function. These coordinates are then used to generate a way points 20 mm above the selected point. Both sets of coordinates are then inputted into the IK algorithm with the specification that the end effector would be oriented orthogonal to the block. The end effector is specified to close on the first click. The outputted joint angles are then fed to the FK algorithm with a 3 second delay between the two points. The user then clicks on a new point in the GUI and the mouse coordinates and depth value are converted to world coordinates. Again, a way-point is generated 20 mm above the goal position, and the end effector is specified to open on the second click. Both coordinates are given to the IK algorithms and the calculated joint angles are fed into the FK algorithm.

III. Results

A. Teach and Repeat

We taught our arm to swap blocks between $(-100, 225)$ and $(100, 225)$ through an intermediate point at $(250, 75)$. The arm was able to cycle the task for at least 8 times. We recorded the joint angles over time for 1 cycle using `ros2 bag`.

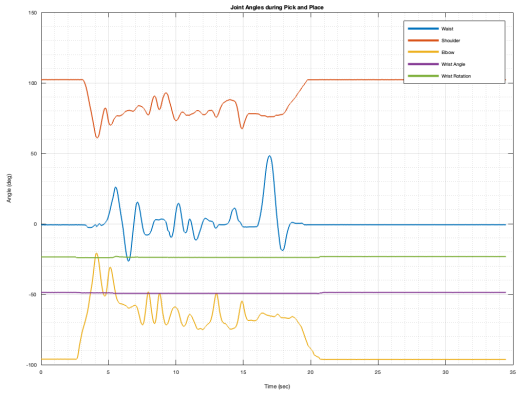

Fig. 6. Joint States of Teach and Repeat Task

B. Camera Calibration

As described in the previous session, two critical matrices are involved for the camera calibration process, namely, the intrinsic matrix K and the extrinsic matrix H. We obtained both 2 different versions for both of them and evaluated their performance based on our system.

The intrinsic matrix was first obtained from the ROS2 topic, `camera_info`, which contains the factory-calibrated intrinsic matrix K_{fac}:

$$K_{fac} = \begin{bmatrix} 900.54 & 0.00 & 655.99 \\ 0.00 & 900.90 & 353.45 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}.$$

Meanwhile, we obtained the camera intrinsic matrix using a checkerboard with ROS camera calibration package. We carried out this manual calibration 4 times and calculated the average value as

$$K_{avg} = \begin{bmatrix} 921.29 & 0.00 & 650.26 \\ 0.00 & 924.10 & 354.35 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}.$$

Compared to the factory-calibrated one, the x and y focal lengths are both larger by 2.22% while the principal offsets are almost the same (differences $< 1\%$). The factory-calibrated intrinsic matrix is generally reliable since it has been through tests and should be good enough for most of the tasks. But the manually calibrated one is more real-time and are meant for the very specific camera piece, i.e., the one we use, which helps avoid the influence of drift. We found out that when we fed the manually calibrated one to the `solvePnP` function, we got better result.

The extrinsic matrix was first obtained from physical measurements using caliper and a tape measure. The angle of inclination was calculated using the

accelerometer data from `realsense-viewer` program. Combined both we got:

$$H_{naive} = \begin{bmatrix} 1.00 & 0.00 & 0.00 & 10.00 \\ 0.00 & -0.99 & 0.10 & 130.00 \\ 0.00 & 0.10 & -0.99 & 1025.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

At the same time, the one we obtained from `OpenCV` using `solvePnP` was

$$H_{cv} = \begin{bmatrix} 1.00 & 0.00 & -0.01 & 20.96 \\ 0.00 & -0.99 & 0.11 & 142.27 \\ -0.01 & -0.11 & -0.99 & 1036.77 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

We noticed that the rotational matrices are almost the same while translation are approximately $10mm$ off for each dimension. We made assumption that the camera is tilted only with respect to the x-axis, which might not be true, and H_{cv} proved it. The error for the translation comes from the physical measurement since we were not sure where exactly the sensors are inside the realsense camera and we used tape, which does not have a very high precision. We were confident regarding to H_{cv} since we fed 16 points to the `solvePnP` function.

To compare and verify the accuracy of these matrices, we tested it by hovering our mouse over the block stacks on the GUI to see if the returning coordinates are reasonable with respect to the true ones in the workspace. We found that using K_{avg} and H_{cv} yielded a better result, which had an average error of $(\pm 1, \pm 1, \pm 3)$ in mm.

C. Workspace Reconstruction

The workspace reconstruction mainly focused on presenting a straight bird-eye view of the workspace on the GUI, which requires a homography transformation for the raw camera view. The homography matrix was calculated using `findHomography` from `OpenCV`:

$$Homography = \begin{bmatrix} 1.366 & -0.073 & -268.917 \\ -0.006 & 1.138 & -51.570 \\ 0.000 & 0.000 & 1.00 \end{bmatrix}$$

After applying the homography matrix, we noticed that there were traceable error for the z values of each location. Thus, we calculated a plane function to correct the errors everywhere in the workspace:

$$z_{offset} = \frac{1194 \times x_{val} + 2498 \times y_{val} + 1560276}{149902} \tag{23}$$

Next, we projected grid points from world frame onto our GUI to verify the correctness of the homography matrix.

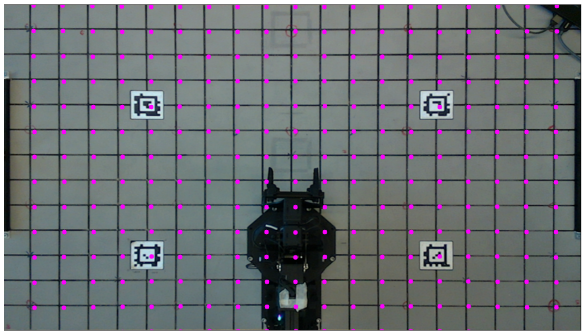

Fig. 7. Output of Project_Grid_Points Function on GUI

The points are projected on the target locations with acceptable errors. Thus, we verified the correctness of the workspace reconstruction method, which should be reliable for further block detection.

D. Block Detection

The block detection algorithm has been tuned towards different lighting conditions. We tested it each time before carrying out a task where the test returns the colors of the blocks and the size of them based on the segmentation areas.

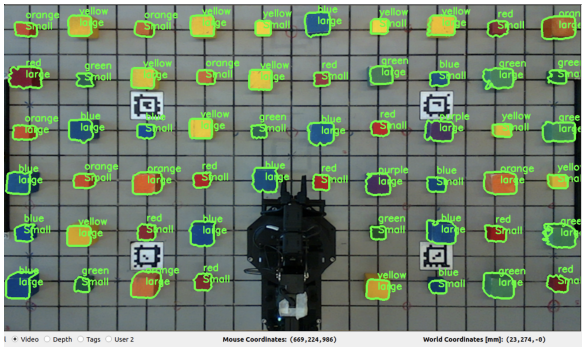

Fig. 8. Implementation of Block Detection Algorithm

Overall, it is reasonably accurate and is sufficient for the further task operation. We generated a heatmap to illustrated the accuracy of our block detection algorithm at the final stage. Approximately 95% of the location in the workspace have a euclidean error less than $8\ mm$. We also calculated the Root Mean Square Error and standard deviation for this test as shown in TABLE I.

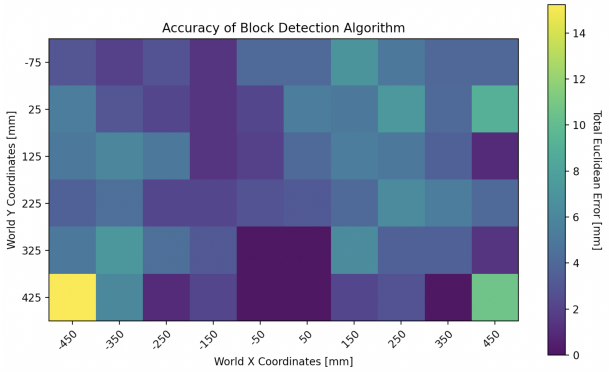

Fig. 9. Accuracy of Block Detection Algorithm

TABLE I
RMSE & STANDARD DEVIATION OF BLOCK DETECTION

Dimension	RMSE	Std. Dev.
X [mm]	4.19	3.39
Y [mm]	2.17	2.10
Z [mm]	5.00	2.90

IV. DISCUSSION

A. Task 1 - Pick 'n sort

In this first task, we achieved a total score of 300. We had to place the small and large box on the third and fourth quadrant of the layout, from first and second quadrant. We employed the strategy of picking up the blocks that are radially close and then move towards blocks that are radially far away. This helps us ensure that when the robot is trying to grab a block that is far off in a horizontal position, the link 2 does not get stopped from reaching its position from a radially closer block. This is the strategy that we employ in future tasks as well. After picking up the blocks, we move them to a pre-specified coordinates based on the size of the block. In addition we optimized the speed and acceleration of the arm in order to finish the task in the given amount of time.

B. Task 2 - Pick 'n stack

In this task, we achieved a total score of 220. We had to place 3 small and 3 large box on two of the AprilTags. Similar to the previous strategy we picked up the radially closer blocks first, however, we picked up large blocks first to act as a large base for stacking future small blocks. We lost our points due to small inaccuracy in the placement of second and third small blocks on a large box. Because the margin of error is less for small box, a small inaccuracy led to the toppling of the top box on the AprilTag. We believe that with a little adjustments in the handling of inaccuracies, we could have achieved the perfect score.

C. Task 3 - Line 'em up

In this task, we achieved a total of 380 points. We had to place 6 big blocks and 6 small blocks in a straight line according to the color order of ROYGBV. We selected a predefined position to place our blocks in the line. To handle the cases, where there are blocks placed on these predefined positions in the initial configuration, we first remove any blocks near the predefined position and place them at AprilTag positions, since we assume there are no blocks at these positions. And then we start lining the blocks according to the color order in these predefined positions.

D. Task 4 - Stack 'em high

In this task, we achieved a total of 260 points. We completed the first level seamlessly which consisted of picking up 6 of the big blocks in rainbow order and stacking them on the bottom right AprilTag. In our attempt of level two, we were able to stack the 6 big blocks in rainbow order, but only could stack one of the small blocks on the bottom left AprilTag. For this task, our strategy consisted of picking up the big and small blocks in alternating order. In this sense, the arm attempted to grasp the big red block and transport it to the right AprilTag, then it attempted to grasp the small red block and transport it to the left AprilTag. For every grasping position, we opted to orient the end effector parallel to the blocks. This was chosen due to a bug in our code which enabled the gripper to grasp a block in the parallel position, but then would switch to the orthogonal orientation when placing the block. This resulted in the block falling off the stack because block's edge would be facing the top of a stack rather than a flat face. To compensate for this bug, the decision to solely grasp in the parallel manner was made on the day of the competition. We only lost points with the small blocks due to the small inaccuracies in our predefined locations for the small blocks to be placed. Our algorithm could grab every block, but the blocks would fall of the stack because the location coordinates were incorrect. Small adjustments to the final locations, or utilizing the RGB camera to define a placement location relative to the previous block could have compensated for this error.

E. Bonus Event - To the sky

In this task, we achieved a total of 1200 points by stacking 15 blocks. As seen in our report, our block

detection, IK, and FK algorithms were designed well but contain subtle inaccuracies that affect the control of the arm. This task required precision and reliability, so we decided to hard code all 15 placement positions of the blocks. Additionally, every block we stacked was placed in the same starting position. In this manner, we also hard coded the initial grasping position of the end effector so that every block was grabbed in the same orientation. The algorithm was split into two parts, the initial 11 block stack and the final 4 block stack. For the first 11 blocks, we relied on our IK and FK algorithms to stack the blocks at the coordinate (0, 300). After building the initial stack, we built a 5 block stack at the coordinate (475, -35). Finally, the the top 4 blocks of this stack was transferred to the top of the initial stack. This task illustrated the constraints of our FK and IK algorithms. Our group found that our algorithm would fail if a 12th block was attempted to be placed at the top of the initial stack. Thus, we sought to build a smaller stack which could be transported. However, this also posed a problem because the additional weight of carrying multiple blocks affected our FK algorithm. The arm would attempt to move to the desired joint angles; however, the added weight would caused the shoulder and elbow joints to droop, affecting the end effector coordinates. In our placement algorithm, we added angle offsets to compensate for the misalignment.

F. Improvements

We believe that there is a room for lot of improvements in both the motion planning and task specific algorithms. In particular, in the motion planning algorithm we have to handle the case where the end-effector moves from a horizontal position to the orthogonal position, when going from a waypoint to grasping position at certain positions on the board. The problem arises when the arm cannot the waypoint with the end-effector in the orthogonal position, but is able to reach the grasping position in orthogonal position, so it has to move from a horizontal position to orthogonal position while grasping, in the course of which it might move the block.

We tried to handle the situation by first increasing the z-distance of the waypoint to give more room for the end-effector when moving from horizontal to orthogonal position and also reducing the speed when moving from these particular waypoints to the grasping position. With these strategies we were able to handle almost all the cases, but in a rare event the end-effector pushes the blocks and ends up either not grasping or grasping at a wrong position. We believe a better way to handle this case would be raise a flag when this situation arises and change the waypoint and trajectory of the path for this particular case. However, due to the constrain on the time, we left the handling of this rare event.

On the event tasks, we believe that the current algorithms can be significantly improved in terms of robustness. Again due to the constrain of time, our code is not robust enough to handle unforeseen scenarios, but with some thoughts most of the unforeseen scenarios can be handled. One example would be taking pose of the blocks at each frame and updating our data structure to represent the most current position of the blocks and segment the blocks based on whether they have been handled or not. Another example would be to check if the robot failed to grasp a block using the end-effector coordinates and camera image to verify and retry while handling the cause of error.

V. Conclusion

In this report, we have systematically outlined the development and execution of a series of methodologies and algorithms for robotic manipulation, emphasizing camera calibration, workspace reconstruction, block detection, and the implementation of both forward and inverse kinematics. The integration of these efforts was tested against a variety of tasks, each designed to challenge and evaluate the system's precision, reliability, and overall performance in replicating and executing predefined tasks within a controlled environment.

The findings from this work underscore the importance of precise calibration and sophisticated kinematic algorithms in robotic manipulation tasks. While the system performed admirably across various tasks, the experience also illuminated the need for improved error handling and adaptability to unexpected scenarios as areas for enhancement. In the future, we will focus on integrating advanced motion planning algorithms to enhance the system's adaptability and efficiency in real-time task execution. This includes developing algorithms capable of dynamically adjusting to unforeseen errors or environmental changes, thereby improving the robustness and reliability of the system.

References

[1] A. Sander and M. Wolfgang, "The rise of robotics," *Bcg perspectives*, vol. 27, 2014.

[2] M. Bartoš, V. Bulej, M. Bohušík, J. Stanček, V. Ivanov, and P. Macek, "An overview of robot applications in automotive industry," *Transportation Research Procedia*, vol. 55, pp. 837–844, 2021.

[3] A. C. Simões, A. Pinto, J. Santos, S. Pinheiro, and D. Romero, "Designing human-robot collaboration (hrc) workspaces in industrial settings: A systematic literature review," *Journal of Manufacturing Systems*, vol. 62, pp. 28–43, 2022.

[4] J. Marks, "How computer vision is changing manufacturing in 2023," Feb 2024. [Online]. Available: https://voxel51.com/blog/how-computer-vision-is-changing-manufacturing-in-2023/

[5] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

VI. Appendices

Here are the DH parameters, we tested it by plotting the end-effector positions VI during the pick and place tusk with known joint angles and it's proved correct.

TABLE II
RX200 Arm DH Table

a	α	d	θ
0.00	-1.57	103.91	1.57
205.73	0.00	0.00	-1.31
200.00	0.00	0.00	1.31
0.00	-1.57	0.00	-1.57
0.00	0.00	175	0.00

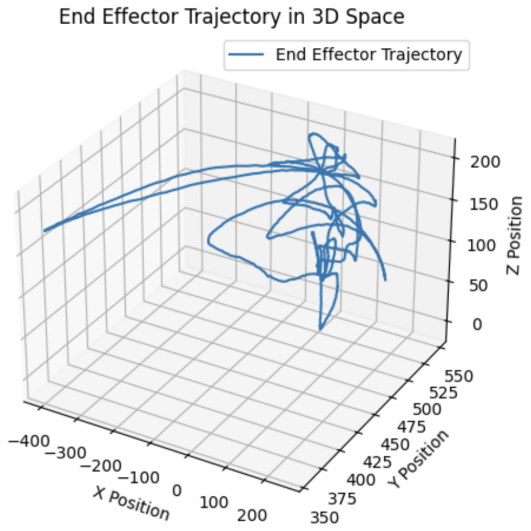

Fig. 10. End-Effector Position for Pick and Place