Autonomous Object Manipulation Using 5-DOF Robotic Arm

Soham Purohit, Saicharan Bandikallu, Andrew Stratton
{sohamsp, saicho, arstr}@umich.edu

***Abstract*—This paper presents a comprehensive study on the autonomous control for a variety of pick and place tasks of a 5-DOF robotic arm, integrated with an RGB and depth camera for computer vision and perception tasks and controlled through the Robot Operating System 2 (ROS2). The presented framework serves as a crucial step toward enhancing the autonomy and versatility of robotic systems in various industrial and research applications.**

I. Introduction

The merging of robotics and computer vision has ushered in a new era of automation and intelligence across various industries. This project aims to use a 5-DOF robotic arm, the RX200, equipped with built-in Dynamixel motors and a Realsense Sensor L515 for computer vision tasks. The control of the robot is carried out using the ROS2 framework. Our primary goal is to enable the robotic arm to recognize and manipulate blocks of different sizes and shapes within its workspace with precision and efficiency, addressing specific task requirements such as picking and sorting them according to size or shape, stacking them on top of each other, and lining them up. This objective involves a comprehensive approach drawing from acting (determining the forward and inverse kinematics of the robot arm), sensing (detecting and determining the world coordinates of blocks of different sizes and colours), and reasoning (path planning).

II. Methodology

A. Computer Vision

1) Camera Calibration: When manipulating objects using a robot taking information from a camera, the first step is to determine the world coordinates of the object from its pixel location on the RGB image obtained from the camera. To do so, we need to determine the camera intrinsic matrix, which transforms the locations of objects in the reference frame of the camera to their pixel coordinates, and the camera extrinsic matrix, which provides the transformation of the world coordinates of the object to the coordinates in the reference frame of the camera. Applying the inverse of the intrinsic matrix on the pixel coordinates followed by the inverse of the extrinsic matrix would allow us to determine the world coordinates of the object from its pixel information.

The intrinsic matrix of a camera depends only on its geometry, the physical characteristics of the sensor, and the details of the image digitization. Hence, it is independent of the factors external to the camera and can be provided by the manufacturer. We used the camera calibration package [1] provided by ROS to estimate the intrinsic matrix and compared it with the factory-provided values to ensure accuracy. This package utilizes a checkerboard of known dimensions placed in several orientations and distances from the camera to determine the calibration parameters for a monocular camera. We performed the calibration several times and took the mean value of the matrix, ensuring good lighting and various orientations and distances from the camera.

Next, we determined the external matrix in two different ways. The first method is by manually calculating the distances of the camera from the origin of the world frame using a measuring tape and constructing the rigid-body transformation matrix. The second method involves using AprilTags [2], a commonly used fiducial system in robotics. We get the extrinsic matrix by matching the exact world coordinates of the detected AprilTags with their pixel locations using OpenCV's [3] Perspective-n-Point (solvePnP) method.

2) Workspace Reconstruction: Now that we have the intrinsic and extrinsic camera matrices, we can reconstruct the world coordinates from their pixel locations. The first equation transforms the pixel coordinates to the camera coordinates, while the second transforms the camera coordinates to the world coordinates. The equations that transform the pixel locations to the world coordinates are-

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = z_c \cdot R_x \cdot K^{-1} \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} ; H^{-1} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

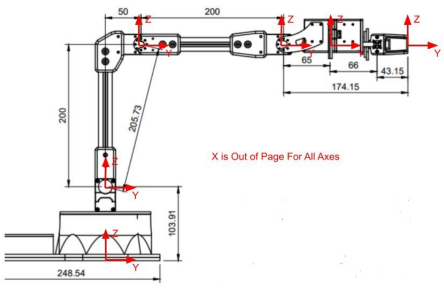

Fig. 1: Axes of each joint

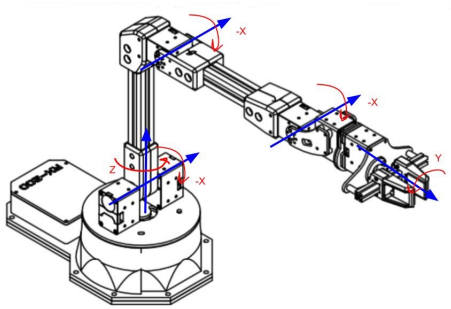

Fig. 2: Direction of rotation of joints

We obtain z_c from the depth sensor. H and K are the extrinsic and intrinsic matrices of the camera, respectively. Additionally, a rotation matrix R_x needed to be applied since the camera had a slight rotation about its own X-axis. The rotation value was determined by calculating the cross-product of the lines joining AprilTags along a horizontal line and AprilTags along a vertical line.
The image of the workspace in the GUI appears as a trapezoid due to the camera's tilt. To correct this, we apply a homography transform to make it a rectangle. This entire procedure was made wholly automated. By simply clicking a button labeled 'Calibrate,' the AprilTags would be detected automatically, and the whole process would be carried out. The GUI displayed the world coordinates based on the mouse cursor's position on the image.

3) Block Detection: To pick and place blocks, they must be detected and localized in the workspace using the RGB-D images from the Realsense L515. Additionally, some downstream automation tasks require color information, so blocks must be detected according to their color. To that end, the RGB image is converted to the Hue-Saturation-Value (HSV) color space and masked according to the minimum and maximum values corresponding to each color (exact thresholds in table I). A median filter with kernel size $ksize = 5$ is then applied to the masked image to reduce spurious detections.

The contours (set of pixels corresponding to the edges of shapes) in the masked and filtered image are then found using the OpenCV $findContours$ function. A bounding box and length, width, and orientation information (all in pixel space) were then obtained using the OpenCV function $minAreaRect$ on each detected contour. To further reduce spurious detections, rectangles with $300 < length \times width < 3500$ and $0.65 < length/width < 2.0$ are retained, where the thresholds were again manually tuned.

The world coordinates of the remaining rectangles are then found using the procedure in section II-A2 on the mean pixel of each contour. To filter any remaining detections that are outside the arm's workspace, block detections with $-500 < x < 500, -175 < y < 475, z > 500$ are removed, where (x, y, z) are the world coordinates.

The detected blocks are stored as a dictionary for downstream tasks, where each color maps to a list of corresponding block detections.

B. Robot Control

1) Forward Kinematics: We must determine the forward kinematics to get the end-effector pose given 5 joint angles. We used the product of the exponential method due to its simplicity. The first step is to label all the axes for the 5 joints, the home position and the end-effector position. Details of the arm can be found in [2].

After doing so, we could figure out the direction of rotation about each screw axis. The direction of rotation about each joint is labeled as shown in Figure 2

To use the product of exponentials, we need to find the M matrix that indicates the position and orientation of the end effector in terms of the home position. Since all our axes are in the same orientation, the rotation part of the M matrix will be the identity matrix, and the translation part will be a simple offset in the positive z and y directions.

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 424.15 \\ 0 & 0 & 1 & 303.91 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The next step is to find all the screw axes for the 5 joints. The angular velocity component of the screw matrix is simply the axis of rotation, which is labeled in the above diagram. The translation part can then be calculated using the following formula:

	Red	Orange	Yellow	Green	Blue	Violet
Min	(150, 0, 0)	(5, 50, 100)	(15, 100, 100)	(60, 75, 50)	(100, 25, 75)	(110, 50, 50)
Max	(185, 255, 255)	(15, 255, 200)	(35, 255, 255)	(85, 255, 255)	(110, 255, 255)	(150, 255, 255)

TABLE I: Minimum and maximum HSV threshold values used in the object detection pipeline.

$$v = -w \times q$$

Here, q is a point of the axis of rotation concerning the world frame, and w is the angular velocity component we just calculated. Following this procedure will give us the 1x6 screw matrix for each of the 5 axes.

$$S_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$S_2 = \begin{bmatrix} -1 & 0 & 0 & 0 & -103.91 & 0 \end{bmatrix}$$
$$S_3 = \begin{bmatrix} -1 & 0 & 0 & 0 & -303.91 & 50 \end{bmatrix}$$
$$S_4 = \begin{bmatrix} -1 & 0 & 0 & 0 & -303.91 & 250 \end{bmatrix}$$
$$S_5 = \begin{bmatrix} 0 & 1 & 0 & -303.91 & 0 & 0 \end{bmatrix}$$

We can then put each of these screw axes into their equivalent skew-symmetric form following this convention:

$$[S_i] = \begin{bmatrix} [w_i] & v \\ 0 & 0 \end{bmatrix}$$

$$[w_i] = \begin{bmatrix} 0 & -S_i[2] & S_i[1] \\ S_i[2] & 0 & -S_i[0] \\ -S_i[1] & S_i[0] & 0 \end{bmatrix}$$

Once obtained, we can use the product of the exponential formula as follows [4]:

$$T = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} e^{[S_5]\theta_5} M$$

Since the angles θ_1 through θ_5 are given to us, we need to plug in these values alongside the skew-symmetric representation of our screw axis to obtain the end effector position and orientation T in terms of our space/base frame.

2) Inverse Kinematics: In the inverse kinematics problem, we are trying to find the joint angles of the robot arm to reach a specific pose of the end effector. There are two approaches to this problem: the numerical and the analytical. We chose to analyze as it is more computationally efficient and will always guarantee a solution, as it does not depend on a starting guess position. Since our robot has 5 degrees of freedom, we will need to split the joints up to solve for the position or orientation of the end effector. In this case, the first 3 joints will be used to set the robot's wrist center to the desired position, and the last two joints will orient the end effector. First, we must find the wrist center given the end effector position. The wrist center is where the last two joint axes of rotations intersect. In our case, the wrist center is shown in Figure 3.

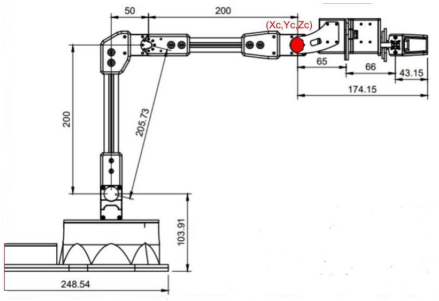

Fig. 3: Position of wrist center

Now that we know where the wrist center should be, we could solve for the exact position by identifying that the position of the wrist center is simply the difference between the desired end effector position and the product of the wrist length and the y-axis rotation of the end effector. We take only the y-axis component since the subtraction should be along the axis of rotation of the last joint, which is the y-axis. The formula is as follows:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} X_{des} \\ Y_{des} \\ Z_{des} \end{bmatrix} - 174.15 * R_{06}[:1]$$

The first angle is:

$$\theta_1 = -\arctan(\frac{x}{y})$$

Given that our θ_1 angle is solved for, we can define a new axis L that is along the direction of that angle such that:

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

We can then draw a plane about the L and Z axis to solve for θ_1 and θ_2, as shown in Figure 4

We can then use the law of sines, the law of cosines, and simple trigonometry to solve for all the intermediate angles and then finally the θ_2 and θ_3 angles. L_1, L_2, L_3

are known:

$$\gamma = \arctan(\frac{L_2}{L_1}) \qquad d_1 = \sqrt{L_1^2 + L_2^2}$$
$$l = \sqrt{x^2 + y^2} \qquad d_3 = \sqrt{z^2 + l^2}$$
$$\alpha_1 = \arctan(\frac{z}{l}) \qquad \alpha_5 = \arctan(\frac{l}{z})$$
$$\alpha_2 = \frac{\pi}{2} - \alpha_1 \qquad \alpha_9 = \arctan(\frac{L_1}{L_2})$$
$$d_2^2 = d_3^2 + 103.91^2 - 2 * d_3 * 103.91 \cos(\alpha_2)$$
$$\frac{\sin(\alpha_3)}{d_3} = \frac{\sin(\alpha_2)}{d_2} = \frac{\sin(\alpha_4)}{103.91}$$
$$\alpha_6 = \arccos\left(\frac{L_3^2 - d_1^2 - d_2^2}{-2d_1 d_2}\right)$$
$$\frac{\sin(\alpha_6)}{L_3} = \frac{\sin(\alpha_8)}{d_2} = \frac{\sin(\alpha_7)}{d_1}$$
$$\pi = \theta_3 + \alpha_8 + \alpha_9$$
$$\pi = \theta_2 + \gamma + \alpha_3 + \alpha_6$$

Once we find θ_1 to θ_3. We can plug these values into our forward kinematics equation found previously to find the end effector orientation at the wrist center. We can use this and our desired orientation to get the rotation matrix R_{36}. Once we find this, we can take the Euler angles from the rotation matrix to get the θ_4 and θ_5 values to reach the desired orientation of the end effector:

$$R_{36} = R_{03}^{-1} R_{06}$$

Once the R_{36} rotation matrix is obtained, we can extract the Euler angles to get the rotation about the y-axis and x-axis needed to get to the desired end effector orientation. The rotation about the -x-axis is the θ_4 angle, and the rotation about the y-axis is the θ_5 angle. We only decided to have the end effector at a $0°$ or $90°$ orientation since the only time we will require an angle in between is when we reach over a block to pick up another block. We decided that this was a rarity in the tasks we were trying to achieve. We implemented it such that the robot will always try to initially plan to reach the specified waypoint at a $90°$ orientation. If the robot cannot reach that configuration (absolute value inside $\arccos$ is greater than 1), we will approach the block at a $0°$ orientation.

C. Automation Tasks

1) Teach and Repeat: In this task, we had to teach the robot to swap the positions of two blocks using an intermediate location. This task involved two steps. In the first "Teach" step, we enabled a button to record and write the current joint coordinates of the robot to a file. We turned off the torque in the robot and manually moved the arm to a set of waypoints that were desired for us to swap two blocks using an intermediate location in the workspace. At each waypoint, we clicked the button that would append to the list of joint coordinates in the text file. Once we were satisfied with the recorded waypoints, initializing the arm and clicking another button would lead to the robot reading from the text file and moving to the respective waypoint consecutively. This was the "Repeat" step. We made the move time proportional to the maximum absolute change in a joint angle. The acceleration time was set to 16.67% of the move time. We used this procedure to set move times in all other automation tasks. However, this sometimes led to a very rapid motion in the arm; hence, we clipped the move time to have a minimum value of 2s.

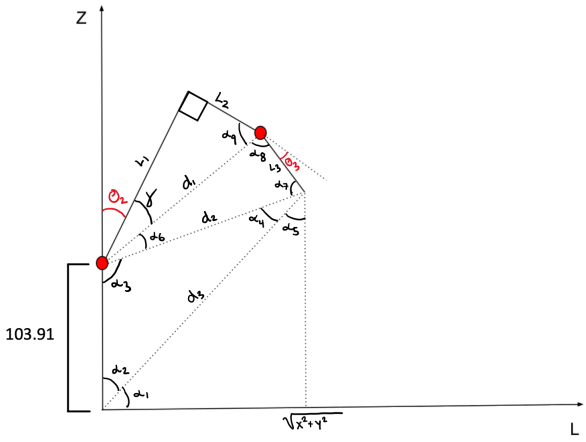

Fig. 4: Inverse kinematics diagram

2) Click to Grab/Drop: In this event, we picked up and placed blocks using mouse clicks on the GUI. For this, we recorded the world coordinates of the mouse cursor when clicked, performed inverse kinematics to determine the corresponding joint angles, and made the robot move to those joint angles. We set two waypoints above the position trying to be reached so that the robot would not collide with the object it was trying to pick up. It would first reach the waypoint directly above, then slowly move down to the desired position to pick up or place the block, and then move back to the first waypoint above the desired position. This method was followed in all future tasks as well. The gripper state was toggled, i.e., if it is open currently, it would move to the desired position and close, and vice versa. This allowed us to seamlessly transition between picking up and placing blocks without the need for additional buttons.

3) Pick and Sort: The pick and sort task consists of picking up big blocks and small blocks of any color and placing them on either the right or left side of the negative plane. Small blocks are placed on the left half of the negative plane, and big blocks are placed on the positive plane. The initial blocks will be kept in the

positive half of the plane and may be stacked to a height of two.

Our block detection algorithm gives us a list of Blocks that are part of a Block class we defined. Each block has its color, coordinates in the world frame, and orientation. These are all obtained with our block detection algorithm when calibrating the robot arm.

To complete this task, we looped through all the blocks that we detected as small and identified the coordinates of the blocks. As mentioned in the previous task, we followed the methodology of setting waypoints for the picking-up task. Then, we determined a waypoint on the left side of the negative plane to place the block. We added a small randomly generated noise offset in the x and z direction so that subsequent blocks will not collide with one another when placed. Once we looped through all the small blocks, we looped through the bigger blocks, still calculating the 3 waypoints to pick up, but instead placing it at a waypoint on the negative right side of the workspace with a random noise offset.

Our algorithm continues this process iteratively until there are no more blocks present in the positive half of the workspace. This allows the robot to eventually pick up all blocks if it misses some in the initial run.

4) Pick and stack: The pick and stack task consists of picking up big blocks and small blocks of any color and placing them in two different stacks on top of two different AprilTags. The initial blocks will be kept within the 4 AprilTags on the workspace and may be stacked to a height of two.

For the pick and stack event, we use the same algorithm as the pick and sort, except the place point is now on a specific Apriltag. We got the x and y world coordinates of the first and second AprilTags from our Apriltag detector that we used for our extrinsic camera calibration as the destination points for our small and big blocks, respectively. Our depth camera z value determined the z-coordinate of the destination point. Depending on the stack, we added the height of the small or big block to the z-value of the depth camera. This way, if we missed picking up a block when we were supposed to, the algorithm would continue to place the block at the correct z-value as we would get the real-time z-value at the specific stack location.

5) Line them Up: The line them up task consists of picking up blocks randomly placed in the workspace (possibly stacked) and placing them in two separate lines (one for large blocks and one for small) in order according to color.

To accomplish the task, we first precompute the placement locations for each block size and color as $(x, y+i\times j, 0)$ where $(x, y, 0)$ is the starting location for each line in world coordinates, $i = 60, 42$ for large and small blocks respectively, and j is the index of the color

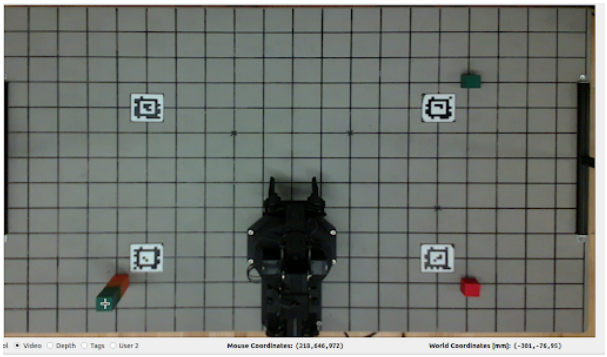

Fig. 5: World coordinates of hovering mouse

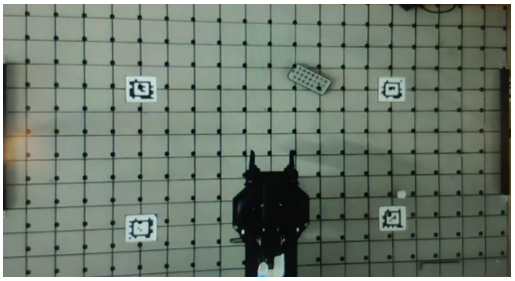

Fig. 6: Projected grid points

in the list $Red, Orange, Yellow, Green, Blue, Violet$ (0-indexed).

The blocks for each color are detected at the algorithm's start and picked up and placed in the order specified in the above list.

6) Stack them high: The stack them high task consists of picking up blocks randomly placed in the workspace (possibly stacked) and placing them in two separate stacks (one for large blocks and one for small) in order according to color.

To accomplish the task, we use the same algorithm as in II-C5, with the only difference being that the placement locations are precomputed as $(x, y, i \times j)$ to reflect the necessary stacking behavior, with $i = 35, 25$.

III. Results

A. Camera Calibration

The manufacturer-provided matrix is-

$$K_{man} = \begin{bmatrix} 902.19 & 0.0 & 662.35 \\ 0.0 & 902.39 & 372.23 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The results of the automated calibration procedure were

$$K_{cal,avg} = \begin{bmatrix} 902.51 & 0.0 & 667.56 \\ 0.0 & 901.08 & 389.11 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

We observe the two matrices' close correspondence. Sources of error in camera calibration may include lens distortions, imprecise measurements of checkerboard dimensions, camera movement during calibration, or errors in the calibration algorithm itself. The factory calibration matrix is a general calibration that the camera manufacturer provides and may not be perfectly tailored to our specific camera's characteristics. On the other hand, the calibration obtained through checkerboard calibration is tailored to your specific camera, considering its unique characteristics and potential lens distortions.

The AprilTag Detections were accurate. This enables us to use the coordinates of the AprilTags to carry out workspace reconstruction accurately. The results of the AprilTag detections were labeled on the GUI, along with the ID of each.

The nominal matrix obtained from physical measurements is-

$$H_{nom} = \begin{bmatrix} 1.0 & 0.0 & 0.0 & 50 \\ 0.0 & -1.0 & 0.0 & 125 \\ 0.0 & 0.0 & -1.0 & 1010 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The extrinsic matrix obtained from solvePnP() is-

$$H_{cal} = \begin{bmatrix} 0.9999 & -0.0090 & -0.0078 & 48.64 \\ -0.0070 & -0.9765 & 0.215 & 122.74 \\ -0.0096 & -0.215 & -0.9765 & 1040.29 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

We observe a close match of the two matrices. There is a slight mismatch in the rotation part of the extrinsic matrix because the camera is not exactly vertical and has slight tilts about its own x and y axes. Further, the translation values may have human error while measuring using a tape measure due to the difficulty in determining the exact location of the camera aperture. Due to these reasons, we decided to use H_{cal} in our workspace reconstruction.

To correct the trapezoidal view to become rectangular, we applied a perspective transform on the centers of the AprilTags that we were automatically detecting. We mapped these centers to the corresponding pixel locations- (378,250), (908,250), (378,524), and (908,524). These values were determined by trial and error, and enough space around the workspace was visible after the transformation to ensure that objects at a height remained visible when placed close to the workspace border. The homography matrix that we obtained after the procedure was -

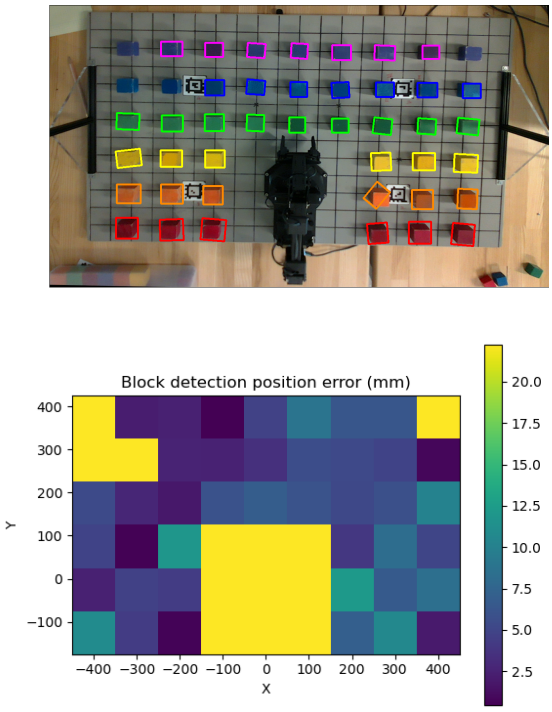

Fig. 7: Block detection evaluation setup and the resulting position error heatmap. The above setup is repeated with each color in each row, and with both large and small blocks.

$$P = \begin{bmatrix} 1.11 & -0.13 & 137.91 \\ 0.03 & 0.90 & -32.17 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

In this matrix, the first row is responsible for the horizontal, while the second row is responsible for the vertical stretching, skewing, and translation of the image. The third row is responsible for the projective aspect of our transformation.

B. Workspace Reconstruction

To test the accuracy of our pixel coordinates to world coordinates transformations, we placed blocks of dimensions 25 mm in stacks of height 0,1,2,4,6 at five different locations in the workspace. These locations were well-distributed to test our reconstruction at all points in the workspace. For all these cases, we got a maximum error of ± 5 mm in any dimension, verifying that our reconstruction was accurate. As an example, we can see that the center of a stack of 4 blocks at (-300,-75) displays world coordinates as (-301,-76,95) on the GUI when the cursor hovers over it in Figure 5. We also projected a grid of points on the GUI that

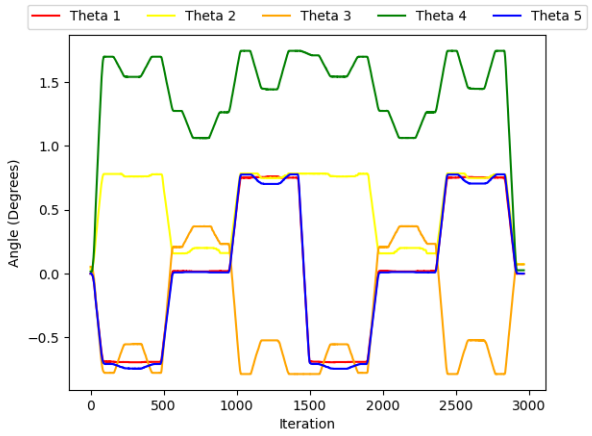

Fig. 8: Joint Angles for Teach-and-Repeat Cycle

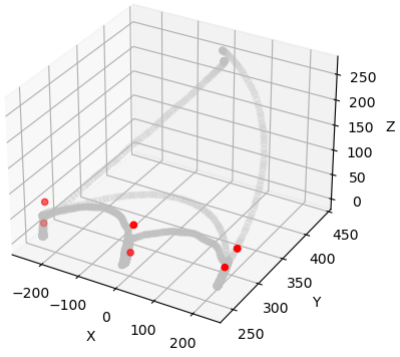

Fig. 9: 3D Position of end-effector in workspace

matches the grid on the workspace base as a second test of our workspace reconstruction, as seen in Figure 6. This projection closely matches the base at all points except in the bottom right corner. These errors can be attributed to the depth sensor performing poorly at points far away from the center of the workspace.

C. Block detection

To evaluate the efficacy of the block detector, small and large blocks of each color were placed in a grid of known x-coordinates = [-400, -300, -200, -100, 0, 100, 200, 300, 400], y-coordinates = [-125, -25, 75, 175, 275, 375], z-coordinates = [0] to serve as ground truth detections. Example evaluation setups and the corresponding error heatmaps are displayed in 7. The average displacement error, maximum displacement error, and successful detections (detections/total GT blocks) overall and by color for each block size are displayed in II.

D. Forward Kinematics

To quantitatively assess the performance of the forward kinematics, we manually moved the arm to five known locations (4 AprilTag locations and a known grid point location) in the workspace and printed out the results of the forward kinematics. We then calculated the norm error between the end effector position's actual and measured values. The results are tabulated in Table **??**. We observe an average error of 14.4 mm in the forward kinematic results.

E. Inverse Kinematics

The analytical inverse kinematics solution can pick up blocks at a 0° and 90° configuration within the configuration space.

F. Teach and Repeat

We cycled through the teach-and-repeat 3 times successfully. The plot of the joint angles for 1 iteration is displayed in Figure 8. We also plotted the end-effector's 3D (x,y,z) position during one cycle. The grey points indicate the trajectory of the robot throughout the iteration. The red dots show the specified waypoint the robot was tasked to reach. This is displayed in Figure 9.

G. Challenge Tasks

1) Pick and Sort: We successfully implemented the pick and sort task, scoring full points in the challenge by correctly sorting all blocks on either side of the robot.

2) Pick and Stack: We were successfully able to stack large blocks on one of the AprilTags. We had issues with the stacking of smaller blocks. This was because of the higher sensitivity required to stack smaller blocks combined with the errors in Forward Kinematics and block detection that led to the toppling of the stack.

3) Line them Up: We were unable to achieve a successful run during the competition time, however with further tuning we were able to line up the majority of the large and small blocks in our own testing afterward. Failure cases still arose due to our algorithm not attempting to redetect and grasp blocks after its initial pass, however each undetected block would not affect others in the line.

4) Stack them High: Similarly, we required further tuning after competition time for this event. We eventually were able to stack four of the large blocks, but were unable to stack small blocks. We attribute the failure to the dependence of higher blocks in the stack on the accurate detection, grasping, and placement of lower blocks, which led kinematics and detection errors to compound for higher blocks. When attempting the task with semicircular and arched blocks, we find we are able to pick and place successfully, but the precomputed

Color	Avg (L)	Max (L)	Detections (L)	Avg (S)	Max (S)	Detections (S)
Red	5.05	16.59	1.00	3.08	7.24	0.80
Orange	3.93	10.52	0.93	3.07	9.40	0.91
Yellow	4.60	12.10	0.91	4.28	9.63	0.91
Green	4.52	15.38	1.00	3.97	18.11	0.91
Blue	4.10	12.26	0.88	4.26	13.23	0.88
Violet	4.78	10.93	0.95	3.06	8.69	0.82
Overall	4.50	16.59	0.94	3.54	18.11	0.87

TABLE II: Block detector position error by color, block size, and overall.

True Value	Measured Value	Norm Error
$(-250, -25, 0)$	$(-242, -30, 0.12)$	9.43
$(250, -25, 0)$	$(240, -15, 3)$	14.45
$(0, 175, 0)$	$(0, 159, 3)$	16.3
$(250, 275, 0)$	$(238, 273, -9)$	15.15
$(-250, 275, 0)$	$(-254, 260, -6)$	16.64

TABLE III: Normed Errors in Forward Kinematics

heights for placing the blocks were no longer valid, so we were unable to stack them without changing our algorithm.

IV. Discussion

Some regions of the workspace had errors in the reconstruction; this led to unsuccessful outcomes when blocks were placed in the workspace's bottom right or bottom left corner due to the inaccuracy of the depth camera away from the center of the board. The limitations created by the inaccuracy of the forward kinematics led to issues in the stacking tasks, especially for the $25mm$ small blocks. We can fix this by considering joint offsets in the initial position that we can optimize through testing. The inverse kinematics method of using either 90° or 0° wrist angles based on the desired position was sometimes limited, as the robot picking up blocks in the 0° configuration would ram into the sideboards. A solution to this can be a better choice of adding waypoints between the picking and placing tasks, leading to smoother paths and preventing collisions.

As noted in II, the block detections are slightly inaccurate overall, with slight variations from color to color. We find that small blocks have a similar magnitude of error in detection to larger blocks while also being detected less frequently. This, combined with kinematics errors, made accurately picking up and placing small blocks significantly more challenging than large ones. These errors, in particular, led to degraded performance on the two stacking tasks, where errors in the placement of blocks lower in the stack would affect the success of higher block placements. In particular, small red blocks were challenging for our system to detect, which are the bottom block on which all future block placements depend in the small stack for 'Stack them high,' which made the detection failures particularly problematic on that task.

V. Conclusion

In this report we have discussed the methodologies we have used to build a robot capable of performing a series of pick and place tasks. We have used computer vision techniques to detect blocks in a specified workspace, camera calibration to convert these block pixel locations to world coordinates, and forward and inverse kinematics in order to move the robot to the appropriate pose. We have discussed and analysed the results of our methodologies against a few competition tasks and lastly proposed potential improvents to our methods.

References

[1] (2023) General tutorials - camera calibration. [Online]. Available: https://navigation.ros.org/tutorials/docs/camera_calibration.html

[2] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.

[3] Opencv documentation. [Online]. Available: https://docs.opencv.org/4.x/

[4] K. M. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*, 1st ed. USA: Cambridge University Press, 2017.

[5] Rx200 arm drawings. [Online]. Available: https://www.trossenrobotics.com/reactorx-200-robot-arm.aspx#drawings