ROB 550 ArmLab Report - AM Group 9

Adam Cherepon
Master's in Mechanical Engineering
University of Michigan
Ann Arbor, Michigan
adamcher@umich.edu

Ziyu (James) Wang
Master's in Robotics
University of Michigan
Ann Arbor, Michigan
jameswzy@umich.edu

Yuzhou (Joe) Chen
Master's in EECS and ME
University of Michigan
Ann Arbor, Michigan
yzc@umich.edu

***Abstract*—The ability of robots to operate autonomously in real-world environments hinges on their path planning, control, and perception capabilities. These functions demand robust algorithms for motion control and sophisticated sensing to properly detect, reason, and act based on observations within their operational space. This approach integrates an advanced computer vision algorithm and a comprehensive path planning strategy to enhance the autonomy of the RX200 robotic arm in manipulating objects with precision. Employing a RealSense L515 RGB-D camera, color and depth images were obtained to sense and analyze the environment, focusing on the detection of wooden blocks by their 3D positions, orientation, color, and size. This system processes these images using OpenCV tools and employs the D-H table for forward kinematics, along with a geometry-based closed-form solution for inverse kinematics. This combination allows the robotic arm to accurately identify and manipulate objects of various sizes, shapes, and colors within its reachable workspace. Despite its success in competitive lab settings, this system requires further refinement to address challenges such as incorrect gripper orientation and to enhance the robustness of the motion vision algorithm for more complex manipulations. These methodical advancements contribute significantly to the field, providing a scalable solution for robotic arms tasked with precise and adaptive operations in dynamic environments.**

Index Terms—Robotics, Computer Vision, Kinematics, Path Planning, Autonomous Systems, Motion Control, Sensing Technology.

I. Introduction

Robotic manipulators play a crucial role in various industries, including warehouse automation and invasive surgeries. These services are completed by performing assigned tasks and adapting to the surrounding environments, which reduces human-related errors. The purpose of this report is to provide an overview of the key components and experiments involving the robotic manipulator RX-200 arm and the Intel Realsense LiDAR Camera L515. This system encompasses three aspects of robotic control: acting, planning, and sensing. The RX-200 is a 5-Degree of Freedom (DOF) manipulator comprising of a waist, shoulder, elbow, wrist dip, and wrist rotation. The camera subsystem offers color and depth sensing functions, which are calibrated by using AprilTag markers. The arm can detect, pick-up, move, and place colored ROYGBV blocks. The robotic arm workstation is depicted in **Fig. 1**.

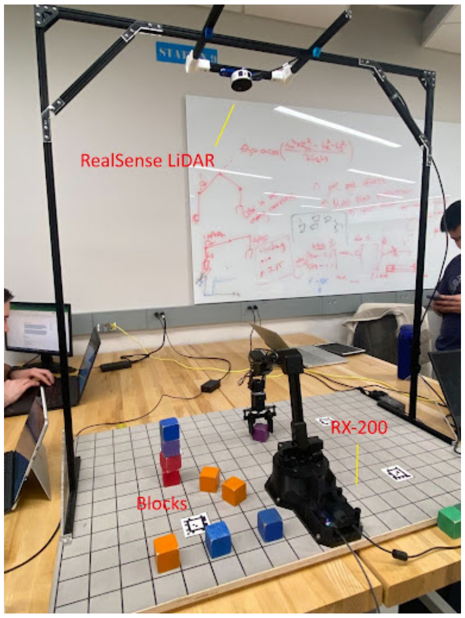

Fig. 1. Robotic Arm Workstation

II. Methodology

A. Camera Calibration

1) Introduction: For the camera calibration, one needs to know the exact coordinate's of the world frame, which is the real and simultaneous location, to direct the locaiton of the end-effector of the robot arm.

2) Pixel coordinate to camera coordinate: As the first step for the camera calibration is to transform the pixel coordinate to camera coordinate, one needs to know the principle of image generation, as shown in **Fig. 2**. The pinhole model shows how the focal length and the offset of the camera location in both the x- and y-axis could affect the raw image.

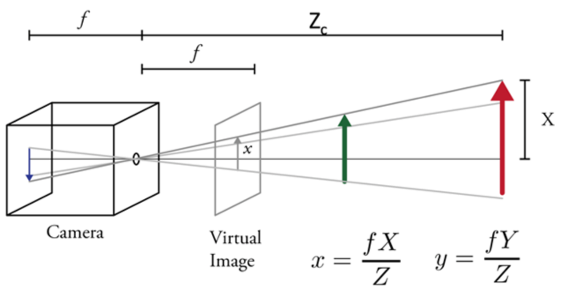

Fig. 2. Pinhole Camera Model

Next, you can introduce the intrinsic matrix for this process, as seen in equation (1).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{I} \quad \overset{projection}{|} \quad 0] \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1}$$

There are several parameters for this equation. α is the focal length in the x-axis. β is the focal length in the y-axis. s is the axis skew. u_0 and v_0 are the principal offsets in the x- and y-axis, respectively.

3) Camera coordinate to world coordinate: As can be seen from **Fig. 3**, the camera is not perfectly parallel to the work station, which will lead to depth error for the camera calibration. This means that pixels far away from the camera but close to the robot arm will have higher depth coordinates than expected. Also, the pixels on the opposite side will have lower depth values.

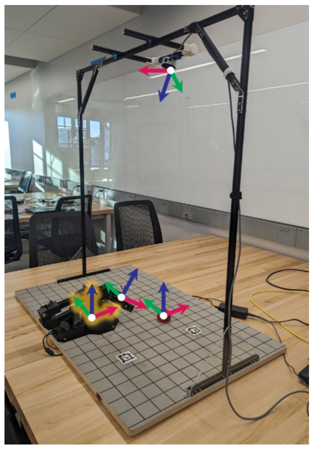

Fig. 3. Image of Work Station with Coordinate Frames

This was solved by introducing the extrinsic function, as seen in equation (2).

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} & \mathbf{R} & & \mathbf{T} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{2}$$

In this matrix, there are two parameters that need to be extracted from the camera information. The first one is **R**, which is the rotation matrix obtained from rotating the camera into the correct position to align it with the parallel of the board. The second one is **T**, which is the location from the board to the camera. After applying this matrix, all the camera coordinates can be represented by the world coordinates.

4) Homography transformation: As can be seen in **Fig. 4**, the image shows a trapezoidal shape, which is incorrect as the original shape is a rectangle.

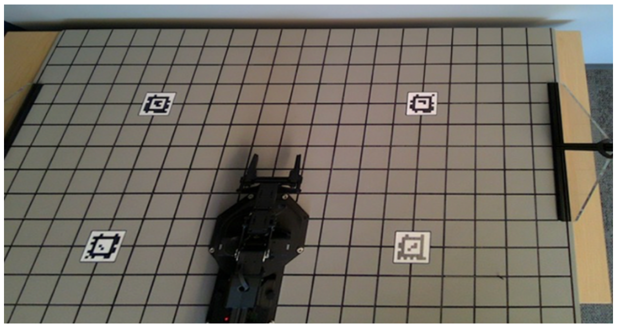

Fig. 4. Trapezoid Image of Work Station from the Camera

To correctly solve this problem and transform that to present the corrected rectangular work station into the image, the projective transformation is introduced, as seen in equation (3).

$$\begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} u_0 \\ v_0 \\ 1 \end{bmatrix} \tag{3}$$

The homography matrix is **H**. To find this matrix, you use the location of the four Apirltags as u_0 and v_0, and the location of their desired pixel coordinates as u' and v' to find the homography matrix **H** using function ***cv2.getPerspectiveTransform()***. Then one needs to apply the transformation to all pixels using the function ***cv2.warpPerspective()***.

B. Forward Kinematics

Forward kinematics is provided the inputs of the joint angles in the configuration space represented by the equation:

$$\mathbf{q} = [\theta_1, \theta_2, \dots, \theta_n]$$

With these angles, the end effector position is found:

$$\mathbf{x} = [x \; y \; z \; \theta \; \psi \; \phi]$$

The Denavit-Hartenberg (DH) Convention was used to create a table and calculate the homogeneous transformations (HT) required to achieve the end-effector position, provided the joint angles. The base frame (frame 0) was assigned on the axis of joint 1, with the z-axis pointing upwards, along the direction of the joint. Frames 1 through 5 were determined by placing them initially at each joint assigned in **Fig. 5**.

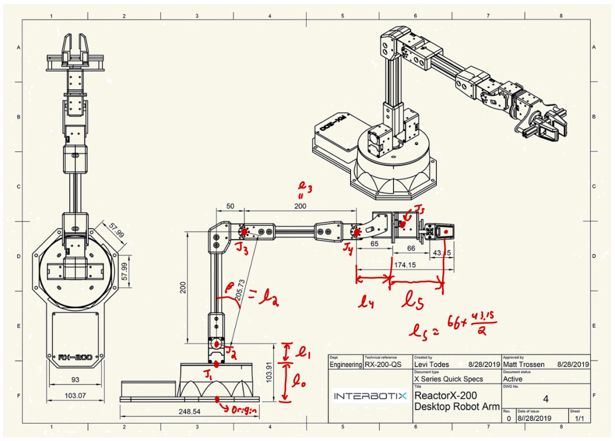
Fig. 5. Interbotix RX-200 Arm Schematic with Annotations

The direction of the z-axes were confirmed when the torque was turned off for the arm and the joints were moved. Once the direction of the z-axes were confirmed, the DH-process was used to determine the directions of the x- and y-axes, and the DH-table was able to be completed. Free-body diagrams (FBD's) were drawn at each joint, and used to determine the orientation of the next axis in relation to the current one. Going from left to right in the DH-table, the order of operations to go from one axis to the next was a rotation about the z-axis, i.e. joint angle (θ_i), translation along the z-axis, i.e. joint offset (d_i), translation along the x-axis, i.e. link length (a_i), and rotation about the x-axis, i.e. link twist (α_i). Filling out the first row of the DH-table, one starts from the base frame orientation and notices the orientation of the z-axis for frame 1. To achieve this new orientation, one would rotate about the z-axis -90°, then translate along this axis a distance of l_0 + l_1 (103.91mm), and then rotate about the current x-axis 90°. This fills out the first row of the DH-table, while noting that there's no translation along the x-axis, so the link-length value is 0 for this row. Going from link 1 to link 2 was more complicated as there's translation in two directions, due to the "L"-shaped geometry of the arm. The solution to this was to align the x-axis with the hypotenuse of the "L"-shape, and translate along that to move from link 1 to link 2. This meant the joint angle was 90°+ the inner angle of the triangle (β), which was found with:

$$\beta = \tan^{-1}(\frac{50}{200})$$

With this information, the joint angle for link 2 is $\theta_2^* + \pi/2 + \beta$, with θ_2^* representing the change of joint 2 from 0. Since there's no translation along the z-axis, the joint offset is 0. The translation along the x-axis is simply the length of the hypotenuse (205.73mm) because of the new alignment of the x-axis with the hypotenuse. Since the z-axis from frame 1 is pointed into the page along with the z-axis in frame 2, there's no need for link twist. To move from link 2 to link 3, the rotation about the z-axis is the remainder of the angle of β from the vertical to align the x-axis along the horizontal, resulting in the value of $\theta_3^* + \pi/2 - \beta$. Knowing that the z-axes are both into the page for frames 2 and 3, the joint offset and link twist are both 0, whereas the link length (translation along the x-axis) is simply l_3 (200mm). Moving from link 3 to 4 requires accounting for the final position of frame 5, as the z-axis needs to point out of the wrist as seen in **Fig. 6**.

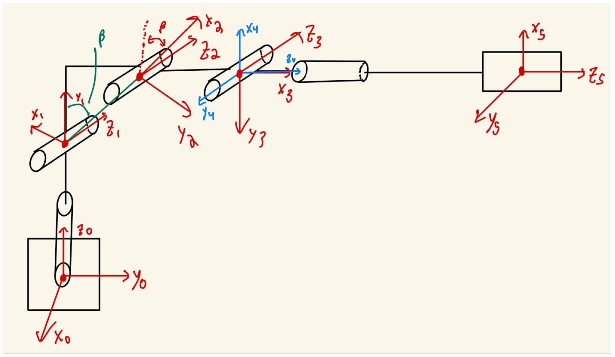

Fig. 6. Arm Kinematics and Frame's.

To achieve this alignment, no translation is required between frames 3 and 4, leaving the joint offset and link lengths to be 0. Then rotating about the z-axis by -90°and the x-axis by -90°, which results in the z-axis pointing in the right direction for frame 5. The last row of the DH-table can now be completed, knowing that there's no link length or link twist, resulting in the joint angle being θ_5^* and the joint offset being the distance between joints 3 and 5 (65mm + 66mm + $\frac{43.15}{2}$mm). The resulting DH-table should look like **Table I**.

Link	θ_i	d_i	a_i (r_i)	α_i
1	$\theta_1^* - \frac{\pi}{2}$	$l_0 + l_1$	0	$\frac{\pi}{2}$
2	$\theta_2^* + \frac{\pi}{2} + \beta$	0	l_2	0
3	$\theta_3^* + \frac{\pi}{2} - \beta$	0	l_3	0
4	$\theta_4^* - \frac{\pi}{2}$	0	0	$-\frac{\pi}{2}$
5	θ_5^*	$l_4 + l_5$	0	0

TABLE I
COMPLETED DH.

C. *Inverse Kinematics*

Inverse kinematics (IK) returns the joint angles from the desired end effector (EE) pose. To solve for the IK for the arm, one needs to define what is known and unknown. The position of the end effector is known (x_e, y_e, z_e) along with the lengths of the arm links and the angles between joints $(l_0 + l_1, l_2, l_3, l_4 + l_5, \beta)$. The variables that IK needs to solve for are the joint angles $(\theta_1, \ldots, \theta_5)$. The twist of the wrist (θ_5) can be assumed to be 0 as it changes the alignment of the wrist, but does not effect the position (x_e, y_e, z_e). From here, one can convert into cylindrical coordinates, simplifying this into a planar problem, allowing to solve for the unknown angles $(\theta_1, \ldots, \theta_4)$, as seen in **Fig. 7** [1].

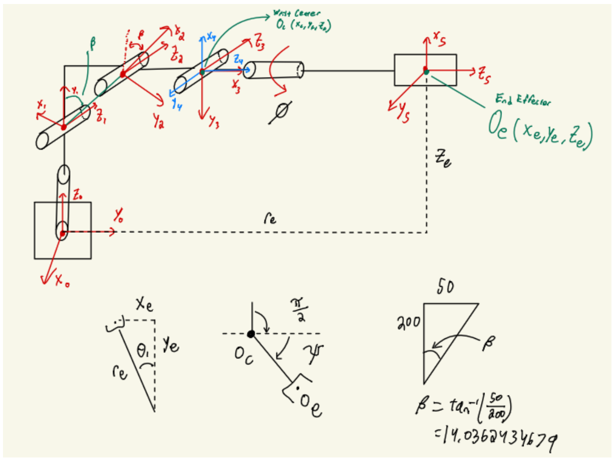

Fig. 7. Wrist and EE Positions and Cylindrical Coordinate Conversion.

Since the x_e and y_e are known, θ_1 can be solved with equations (4) and (5).

$$\theta_1 = \tan^{-1}\left(\frac{-x_e}{y_e}\right) \tag{4}$$

$$\theta_1 = \tan^{-1}\left(\frac{-x_e}{y_e}\right) + \pi \tag{5}$$

The reason there are two solutions is because the base can rotate 180°. Since travel distance is minimized and the arm mostly needs to be within quadrants I and II, equation (5) will not be used in the calculations. Converting to cylindrical coordinates, equation (6) is obtained.

$$r_e = \sqrt{(-x_e)^2 + y_e^2} \tag{6}$$

With r_e and z_e known, r_c and z_c (the r and z for the wrist center) can be found with equations (7) and (8).

$$z_c = z_e + (l_4 + l_5) * sin(\psi) \tag{7}$$

$$r_c = r_e - l_5 * cos(\psi) \tag{8}$$

Assuming the wrist is horizontal in the solution, the value of z_c needs to be modified by subtracting the height of $l_0 + l_1$, resulting in equation (9) for the height coordinate for the wrist center.

$$z_c{}' = z_c - (l_0 + l_1) \tag{9}$$

With these variables, one can substitute into equations (10) and (11). There are two solutions for θ_3 because of the "elbow up" and "elbow down" configurations of the arm, but to avoid collision of the arm with the table, the "elbow up" configuration is the only solution needed.

$$\theta_3' = \pm\cos^{-1}\left(\frac{(r_c^2 + z_c'^2) - l_2^2 - l_3^2}{2 * l_2 * l_3}\right) \tag{10}$$

$$\theta_2' = \text{atan2}(z_c{}', r_c) - \text{atan2}(l_3 * sin(\theta_3'), l_2 + l_3 * cos(\theta_3')) \tag{11}$$

These angles need to be modified to take into account the 50mm offset between l_2 and l_3, as can be seen in **Fig. 8**.

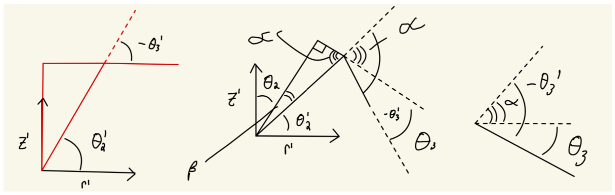

Fig. 8. Schematics of angles θ_2 and θ_3.

From equations (10) and (11) and the drawings in **Fig. 8**, equations (12) [2] and (13) [2] can be derived.

$$\theta_3 = \theta_3' - \alpha \tag{12}$$

$$\theta_2 = \pi - \beta - \theta_2' \tag{13}$$

With all of these variables known, one can solve for θ_4 with equation (14), assuming ψ is known.

$$\theta_4 = \psi - \theta_2 - \theta_3 \tag{14}$$

D. Block Detection

Block detection is essential to isolate the objects (blocks), draw their shapes on the video frame, and label them with size and color. The method is divided into several steps.

1) Convert the image from RGB to HSV: To convert the image from RGB to HSV, it is handled by the OpenCV function ***cv2.cvtColor()***, which is a part of the preprocessing.

2) Isolate the board without robot arm: To isolate the board from the robot arm, one must draw 2 distinct rectangles using the function ***cv2.rectangle()***. The first rectangle is to isolate the board from the surrounding environment. The second is to isolate the robot arm, which means only the image inside the board, excluding the robot arm, can be detected. This step will make the first mask (space mask). Additionally, using depth threshold by setting the minimum height to half the size of the smaller blocks, also needs to be taken into consideration.

3) Isolate the blocks: To isolate the blocks, it can be done by using the HSV threshold, which isolates the contour of the block by using HSV values. The function ***cv2.inRange()*** can easily isolate the blocks by looping the defined color list, which contains 6 distinct thresholds for red, yellow, orange, blue, purple, and green. This step will convert the pixel of blocks to white and others to black. This step will also make the second mask (color mask).

4) Find and draw contours: After the zone and color masks are obtained, one can use the function ***cv2.bitwise_and()*** to combine two masks together, which can isolate the block from both the space and color dimensions. Then one can use ***cv2.findContours()*** to generate the contour list and use ***cv2.drawContours()*** to show the contours on the video frame.

5) Mark blocks with labels: After showing the contours of blocks on the image, one can label them with different sizes and different colors, including 2 sizes ("big" and "small") and 6 colors. This step can be done using the function ***camera.retrieve_area_color()***.

E. Click-to-Grab and Click-to-Drop

The click-to-grab and click-to-drop algorithms control an RX-200 manipulator arm by picking up and placing a block at coordinates selected by users. In the first step, a user selects a coordinate where a block is located on the GUI screen. After this selection, the manipulator arm picks up the block with a specific path planned in ***click_grab()***, mainly when a user clicks the button called 'Click2Grab'. A Boolean variable is defined in ***click_grab()*** to change its value depending on whether a block is picked up or not. If a block is not picked up from the start, the manipulator arm starts picking it up along a series of motions defined in ***move_path()***. Additionally, the ***gripper.grasp()*** method is used to close the gripper to pick up the block. The Boolean variable changes its value to true. In the final motion of picking up a block, the arm moves back to 0 in the x-coordinate. After executing the click-to-grab algorithm, the next arm action is click-to-drop, which involves placing a block by calling the ***click_drop()*** function. This function also calls the ***move_path()*** function for the same motions as ***click_grab()***, but ***gripper.release()*** is used to open the gripper for placing a block.

In addition to the ***click_grab()*** and ***click_drop()*** functions, ***move_path()*** is a major part of the motion plan. This function applies the inverse kinematics function (i.e., ***IK_geometric()***) with inputs of selected coordinates and angle inputs. The first action is to lift the gripper to 275mm in the z-coordinate to avoid colliding with blocks placed on the table. After that, the gripper slowly moves down to 150mm, to also avoid collisions. The final action is letting the gripper move to half the height of the block without touching the table. Once this series of motions is completed, the gripper can be opened or closed for picking/placing blocks.

III. Results

A. Camera Calibration

1) Intrinsic Data: The average intrinsic matrix after performing the checkerboard calibration five times is shown in equation (15).

$$K_{average} = \begin{matrix} [891.57 & 0 & 654.68] \\ [0 & 889.95 & 392.87] \\ [0 & 0 & 1] \end{matrix} \tag{15}$$

The factory calibrated intrinsic matrix is shown in equation (16):

$$K_{factory} = \begin{matrix} [902.57 & 0 & 660.20] \\ [0 & 903.04 & 377.16] \\ [0 & 0 & 1] \end{matrix} \tag{16}$$

Based on these two intrinsic matrices, the points are close to each other. However, there are minor differences by comparing these two matrices. This error could be due to not removing the checkerboard calibration. Another error could be due to objects in the workspace. These possible scenarios may produce an inaccurate intrinsic matrix.

2) Extrinsic Data: The hand-calculated extrinsic matrix is shown in equation (17).

$$H_{hand} = \begin{matrix} [1 & 0 & 0 & 0] \\ [0 & 0.96 & 0 & -350] \\ [0 & 0 & 0.96 & 990] \\ [0 & 0 & 0 & 1] \end{matrix} \tag{17}$$

The program-calculated extrinsic matrix is shown in equation (18):

$$H_{program} = \begin{matrix} [1 & 0.019 & 0.014 & -6.40] \\ [-0.017 & 0.98 & -0.15 & -325] \\ [-0.017 & 0.15 & 0.98 & 953] \\ [0 & 0 & 0 & 1] \end{matrix} \tag{18}$$

By comparing these two extrinsic matrices, the data are slightly different. This means that the calibration is accurate for the purposes of the lab. It also illustrates that the world coordinate measurements are similar in its' accuracy. Also, the program-calculated data works for different camera rotational positions.

3) Homography Matrix: The homography matrix is shown in equation (19).

$$P = \begin{matrix} [1.06 & -0.13 & -51.4] \\ [0.02 & 1.01 & -70.6] \\ [1.90 & -1.67 & 1] \end{matrix} \tag{19}$$

This transformation/projection is done by referencing four fixed AprilTag positions.

4) Calibration Validation: To validate the calibration results, grid points are projected via the video frame. There consists inaccuracies on x- and y-coordinates. Increasing inaccuracies occurred when grid points are expanding from the center to the border of the workspace.

B. Teach and Repeat

The ”teach-and-repeat” task involves manually driving the RX-200 Robot Arm to specific positions and joint angles, while recording waypoints that will teach the robot how to perform actions.

Fig. 9 illustrates the swapping of waypoints and paths as the desired outcome of the ”teach-and-repeat” task.

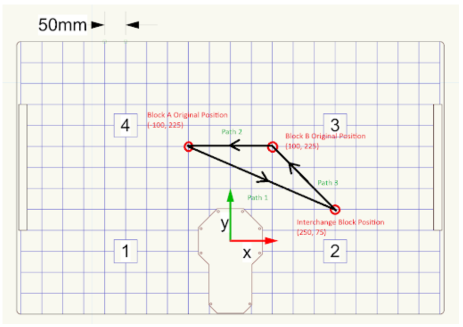
50mm
4
3
1
2
Block A Original Position
(-100, 225)
Path 2
Block B Original Position
(100, 225)
Path 3
Path 1
Interchange Block Position
(250, 75)
y
x

Fig. 9. Teach-and-Repeat motion plan.

There are at least three steps in one cycle of this task. First, the arm picks up and moves Block A to the interchange block position along Path 1. Then, Block B is moved to the original position of Block A along Path 2. The final step is for the arm to pick up Block A from the interchange block position and place it in the initial position of Block B through Path 3. This task can perform one swapping cycle per execution, with uncertainties between the desired coordinates and the actual coordinates for placing and picking up blocks. These uncertainties arise from the arm system not including camera calibration or block detection.

The plot, as shown in **Fig. 10**, displayed five joint angles during one cycle of the teach-and-repeat task over time.

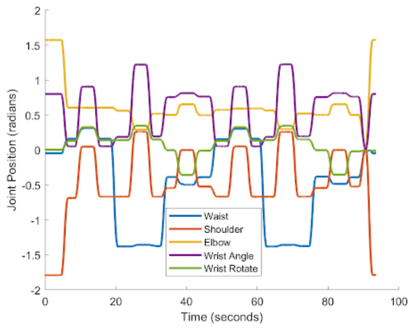

Fig. 10. Joint angles versus time during one-cycle teach-and-repeat task.

C. Forward and Inverse Kinematics

The five joint angles versus time for a teach-and-repeat cycle are shown in **Fig. 10**. These angles were input into the forward kinematics (FK) function to verify the FK.

The inverse kinematics (IK) were verified experimentally by comparing the desired (where the mouse clicked in the camera view in the GUI) to real-world end effector positions, as seen in **Table II**.

	Desired [mm]			**Actual [mm]**		
Locations:	**Xw**	**Yw**	**Zw**	**Xw**	**Yw**	**Zw**
1	100	225	37	91	216	53
2	-150	-75	37	-141	-76	48
3	200	25	37	200	19	50
4	-150	125	37	-141	109	49
5	0	225	37	-4	211	50
6	150	125	37	136	116	53

TABLE II
END EFFECTOR MEASURED AND REAL-WORLD DATA.

These resulting positions were achieved through the geometrical IK solutions that output joint angles. The Z_w coordinates were all offset by 16mm, to prevent collision of the end effector with the table. The error between desired and real-world coordinates, along with the corrected real-world Z_w coordinates are shown in **Table III**.

		Error [mm]		
Locations:	**Corrected Zw**	**Xw**	**Yw**	**Zw**
1	37	9	9	0
2	32	9	1	5
3	34	0	6	3
4	33	9	16	4
5	34	4	14	3
6	37	14	9	0
Average		**7.5**	**9.16667**	**2.5**

TABLE III
CORRECTED END EFFECTOR Z COORDINATES AND ERROR BETWEEN MEASURED AND REAL-WORLD VALUES.

As seen in **Table III**, the error for all coordinates was less than 10mm, which fits within the ± 20mm range that were the parameters.

D. Block Detection

The block detection is shown in **Fig. 11**.

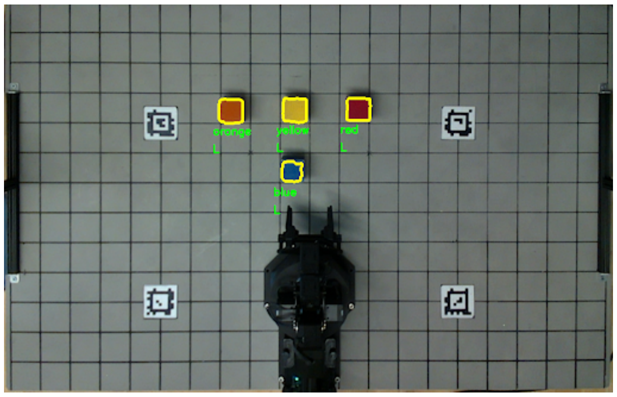

Fig. 11. Calibrated Camera View with Block Detection

1) Convert the image from RGB to HSV: After applying ***cv2.cvtColor()***, a Gaussian Blur was applied, which is the function ***cv2.GaussianBlur()*** in the OpenCV with the 5 x 5 size kernel. This was done to enhance the robustness and stability.

2) Isolate the board without robot arm: To isolate the board from the robot arm, this part was successfully handled by drawing rectangles and making masks.

3) Isolate the blocks: In order to prepare for the competition, the depth mask was removed because it would have been a burden and would have lead to skewed error for the depth, showing the incorrect world frame position after detecting the blocks. The pure color mask was used to isolate the blocks, which included repeated upper and lower bound modification, resulting in the color list shown in **Table IV**.

'red':	(140, 30, 100)	(180, 255, 255)
'orange':	(5, 50, 100)	(15, 255, 255)
'yellow':	(20, 50, 100)	(27, 255, 255)
'green':	(39, 50, 100)	(94, 255, 255)
'blue':	(96, 50, 120)	(130, 255, 255))
'violet':	(110, 50, 100)	(141, 255, 255))

TABLE IV
COLOR LIST.

By using the accurate color list, one can even detect the contours for each block when stacked. To remove noise, the functions ***cv2.erosion()*** and ***cv2.delate()*** were implemented.

4) Find and draw contours: In the implementation of the code, there were some contours notated as single points, which likely resulted from the noise, which needed to be canceled. This led to the use of the function ***cv2.moments()*** to calculate areas of the contours and draw nothing for those with zero contour.

5) Mark blocks with labels: Marking the blocks with labels worked as intended with the code. However, this step could be optimized by labeling the center of the block. This would have provided an advantage during the competition with more accurate grabbing and dropping points.

E. Competition Performance

The competition performance of the robot met basic expectations but fell short of the group's hopes. One of the difficulties encountered was picking blocks up at specific positions during the competition. This issue likely stemmed from problems with the inverse kinematics, leading to the RX-200 arm's failure to pick up a block at these positions and resulting in errors.

However, the arm's performance had positive aspects in the "click-to-grab" and "click-to-drop" tasks. The code was able to manually control the arm to align another set of multiple blocks at different positions on the workspace. Unfortunately, due to aforementioned reasons, not all blocks could be placed in a single line due to constraints in the inverse kinematics.

IV. DISCUSSION

1) Calibration errors: When the "click and grab" part is handled, it is imperative to address the observed limitations in the current automated system. As delineated in the schematic representation **Fig. 12**, optimal performance of the stacking process is contingent upon the absence of lateral or prismatic displacement, which is achievable solely beneath the camera's nadir without deviation.

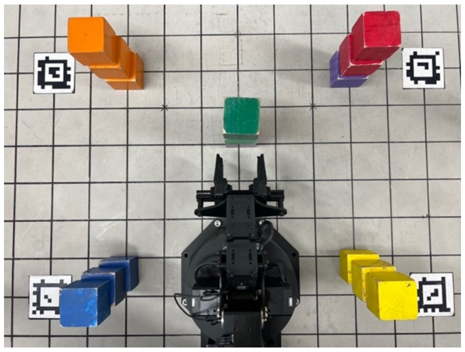

Fig. 12. Camera Image with Stacked Blocks Displaying Slopes.

Furthermore, you could get rid of the ApirlTags after applying the homography transformation the first time.

2) Incorrect Grip: For the incorrect grabbing, as seen in **Fig. 13**, an incorrect wrist alignment was happening during the task. This means that there needs to be an input to the robot arm to align the gripper edges to be parallel with the block, which would correct the grabbing and dropping operations.

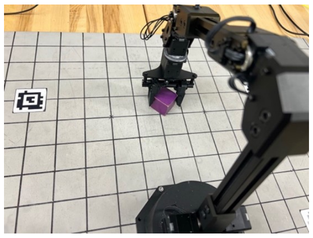

Fig. 13. Example of Incorrect Wrist Orientation in Relation to the Detected block.

3) Inverse Kinematics models: The precision of the inverse kinematics model could be improved, with a targeted reduction of positional error to within 5 millimeters. Furthermore, an offset adjustment for the depth coordinate is essential to prevent overshooting phenomena. Expanding the range of inverse kinematic solutions would allow for a more extensive operational envelope, as seen in **Fig. 14**. An increased range would accommodate higher stacking configurations. The introduction of parallel gripping and releasing actions, as seen in **Fig. 15**, would enhance the system's robustness and stability. This would mitigate risks of overshooting that could compromise the structural integrity of the block assembly.

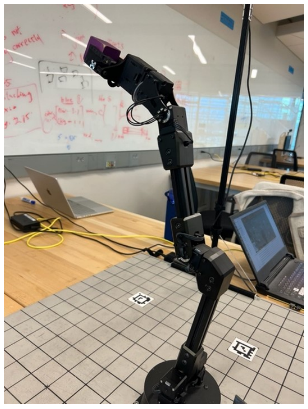

Fig. 14. Example of Increased Range of Arm.

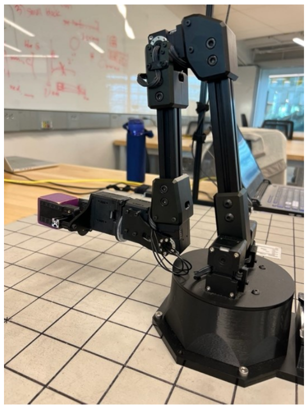

Fig. 15. Example of Gripper Parallel to Workspace.

4) Stack Planning: The strategic planning component encompasses several critical considerations. Foremost, the manipulator arm must be programmed to avoid obstructing the visual sensor's field of view, as the block detection system serves as a navigational guide. Moreover, the integration of an auxiliary stacking model is suggested, whereby a secondary stacking platform ('sub deck') can be positioned atop the primary stack ('main deck'), as can be seen in **Fig. 16**. This strategy has the potential to augment the stacking capacity by an additional eight blocks, thereby optimizing the spatial configuration.

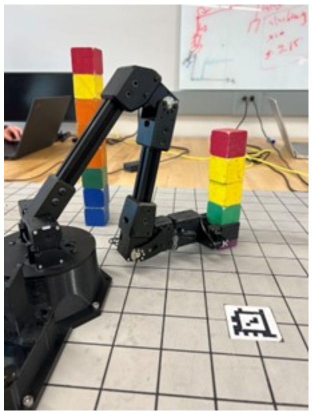

Fig. 16. Example of Pre-planning Stacking.

V. Conclusion

To rectify the spatial inaccuracies in the camera calibration, re-calibration of the depth perception framework is recommended, or alternatively, the introduction of a compensatory algorithmic function designed to neutralize errors. This function would be predicated on the Euclidean distances ascertained from the camera's vantage point.

Concurrently, noise reduction within the visual data can be achieved through the implementation of a secondary homographic transformation between the boundary of the board and desired location. This would be applied to the grid framework, with the objective of rectifying the image geometry.

After these modifications, a re-calibration of the depth frame would be beneficial to enhance the accuracy of the depth mask utilized by the block detection system. Such enhancements are critical for bolstering the system's robustness and stability, which are paramount for competitive performance in automated manipulation tasks.

Improvements to the block detection algorithm are paramount. It is proposed that blocks be annotated with orientation markers, aligning the wrist angle, denoted as ϕ with the corresponding block angle. This alignment is crucial for facilitating seamless engagement of the gripper mechanism. Additionally, the computation of contour centroids should be implemented, enabling autonomous block manipulation by the gripper apparatus.

References

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

[2] K. Lynch and F. C. Park, *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017. [Online]. Available: https://hades.mech.northwestern.edu/images/7/7f/MR.pdf

VI. Appendices

A. Calibration Code

- ***calibrate()*** – calculates the extrinsic matrix with the known AprilTag positions. It also calculator the homography matrix for adjusting its video projection. Reference lines 160-197 in *state_machine.py* for reporting the calibration function.
- ***trackMouse()*** – calculates the world coordinates with respect to the mouse coordinates, the intrinsic matrix, and the extrinsic matrix. It also displays the calculated world coordinates on the GUI. Reference lines 225-267 in *control_station.py*.
- ***projectGridInRGBImage()*** – not only displays grid points on the GUI video frame. It also projects the perspective of the video frame. Reference lines 477-522 in *camera.py*.

B. Forward and Inverse Kinematics

- ***FK_dh ()*** – returns an end effector position of the RX-200 arm with the DH configurations in *rx200_dh.csv*. Reference lines 32-74 in *kinematics.py*.
- ***get_euler_angles_from_T ()*** – returns three euler angles from a 4-by-4 transformation matrix returned from ***get_transfrom_from_dh ()***. Reference lines 105-121 in *kinematics.py*.
- ***IK_geometric ()*** – returns an array of joint configurations that will control the RX-200 arm to the given end effector position. Reference lines 172-231 in *kinematics.py*.

C. State Machine

- ***click_grab ()*** – commands the arm to pick a block with a path defined in ***move_path ()***. Reference lines 366-397 in *state_machine.py*.
- ***click_drop ()*** – commands the arm to place a block with a path defined in ***move_path ()***. Reference lines 404-433 in *state_machine.py*.
- ***move_path ()*** – is the major motion plan function to command the arm to move for picking or placing a block in the workspace. Reference lines 309-322 in *state_machine.py*.