ROB 550 Armlab Report

Trushant Adeshara, Dhyey Manish Rajani, Mujtaba Khan Mohammed
{trushant, drajani, khataba}@umich.edu

***Abstract*—In this research report, we explore the application of robotic manipulators in conjunction with an RGB-D camera for precise object manipulation within a specified workspace. Our study focuses on the utilization of the Interbotix RX200 Arm in combination with the Intelrealsense L515 LiDAR. We provide a comprehensive overview of our software stack development, elucidating the decision-making process that guided our implementation. We touch up on topics including camera calibration, projective transformation, block detection, forward and inverse kinematics.**

I. Introduction

This report introduces the automation of a 5-DoF robotic arm, designed for the purpose of efficiently executing stacking and sorting tasks that involve variously sized and colored blocks. The RX200 robot arm facilitates the actuation, while the RealSense LiDAR camera serves as the sensing component. Homogeneous transformations play a pivotal role in converting pixel and depth coordinates into real-world coordinates, determining the precise location of the blocks for operational purposes. Additionally, fiducials are incorporated for accurate pose estimation.

The control of the robotic arm within the workspace is achieved through the integration of both Forward Kinematics and Inverse Kinematics. These methodologies ensure the precise regulation of the robotic arm to the desired positions, enhancing its efficiency in performing the specified tasks.

II. Methodology

A. Camera Calibration

Every camera requires a calibrated factory matrix specific to its make, that is used as an intrinsic matrix for perspective transformations, as outlined in equation (1).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_C} \begin{bmatrix} \alpha & s & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I|0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1}$$

It transforms image from camera frame to image(sensor) frame. This factory intrinsic matrix proves reliable for computations,as they were calibrated using an ample dataset.

$$\mathrm{K}_{factory} = \begin{bmatrix} 904.571 & 0 & 635.981 \\ 0 & 905.295 & 353.06 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

However, variations in environmental factors, such as different lighting conditions or lens aging, can disturb the camera's performance with the factory-calibrated matrix. In such cases, users have the option to re-calibrate the intrinsic matrix, tailoring it to their unique usage scenario. Our calibration process involved three learning trials using a checkerboard and the ROS camera calibration package.

$$\mathrm{K}_{avg} = \begin{bmatrix} 930.563 & 0 & 629.029 \\ 0 & 933.571 & 368.777 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The data samples for size and skew were different in every trial, which resulted in minute differences among them, but comparing the average of the user calibrated intrinsic matrix from the factory calibrated intrinsic matrix, there were some slight changes. We decided to use given intrinsic matrix since it was extracted using a geometric camera calibration process which is more accurate than single checkerboard calibration.

B. Workspace Reconstruction

One of the important aspect of workspace reconstruction is to go from image coordinates to world coordinates and vice versa. Since, the intrinsics are sorted, the next step is to transform these camera coordinates into world coordinates, as outlined in following equation:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{4}$$

To achieve this, we must compute an extrinsic matrix, which serves as a homogeneous transform matrix, encapsulating the rotation and translation components.

1) Manual Calibration: Our first approach was to manually calculate transformation between the world frame and the camera frame. Our world frame origin was at the base of the robotic arm and the origin for the

camera frame was as close to the centre of the camera lens as possible.Initially, we focused on understanding the rotation between the frames, given their parallel orientation and the z-axis pointing toward each other, indicating a 180-degree rotation in either the x or y axis. In our case, the rotation occurred along the x-axis. Now with that information, it is fairly straightforward to calculate the rotation matrix. For translation, we measured the distances between the origins in all the three axes using a measuring tape, which then gave us the extrinsic matrix as shown in equation below:

$$\mathrm{H}_{manual} = \begin{bmatrix} 1 & 0 & 0 & 28 \\ 0 & -1 & 0 & 182 \\ 0 & 0 & -1 & 1004.2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

However, this method was a rough estimate, resulting in coordinate conversion with discrepancies of around +/- 20 mm. We addressed this by manually calibrating the translation values to bring them closer to actual. Nevertheless, the error rate increased with changes in the depth frame.

2) AprilTags Calibration: Apriltags[1] serve as a widely employed fiducial system in computer vision for calibrating the pose of a system. In our setup, we utilized a total of 4 apriltags.

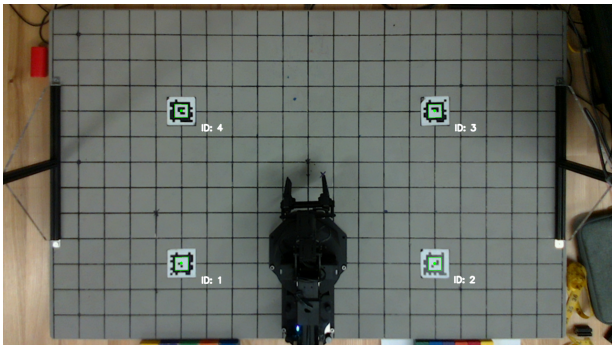

Fig. 1. All 4 apriltags are detected for calibration using ROS

Calibration involved the use of the Perspective and Point (PnP) [2] method, facilitated by the apriltag ROS package. The initial step was to employ apriltag detection to obtain tag coordinates, and the retrieved values proved to be close to accurate. The PnP working requires the inputs of the coordinates of the apriltags acquired from apriltag detection and also their known world frame coordinates. This would give us the extrinsic matrix(6) that will be again used for better than manual perspective transform.

$$\mathrm{H}_{cal} = \begin{bmatrix} 0.999 & -0.035 & 0 & 4.482 \\ -0.034 & -0.992 & 0.118 & 167 \\ -0.015 & -0.118 & -0.992 & 987.356 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

Our initial assumption that the rotation matrix solely involved a 180-degree rotation along the x-axis proved incorrect. In reality, there was a slight rotation along the y-axis as well. Furthermore, the manually calculated translation exhibited considerable errors when compared to the translation values derived through the PnP calibration matrix.

C. AprilTags Homography

One another import usage of AprilTags in our case scenario was to get birds-eye view of the workspace using Homography [3] which is a 8-DoF transform. In order to perform homography we require source points from the base image and destination points of the desired image.

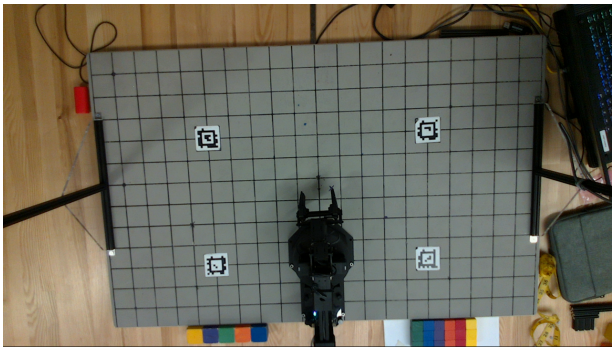

Fig. 2. Image before Homography Transformation

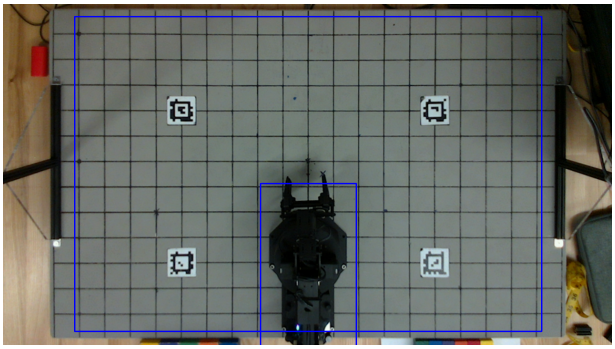

Fig. 3. Bird's eye view after Homography Transformation

D. Block Detection

Our approach for Block Detection includes two steps: Finding contours and deciding color of the block.

Pseudo Algorithm for Block Detection is as Follow:

1) Pre-process depth image with a Gaussian filter of size 5×5.
2) Segment the depth image for contour detection using average thresholding.
3) Find all the contours in segmented depth.
4) For each detected contours we use corresponding HSV image for color classification based on preset range in labels.[4]

5) For the detected contours, we calculated moments which aided in extracting the area of the detected contour.[5]
6) Lastly, a minimum area rectangle was calculated around the detected contours which will provide dimension of blocks to be picked.

If the area(contour) $>$ 300, we would check it if either it is a square, rectangle, parallelogram or circle.

Since the robot is also in the workspace, we used a bitwise mask to permanently remove area around robot and HSV range of color provides an add on safety that robot won't be detected while performing tasks. This can be seen as blue outline in fig. 3.

Output of the block detection algorithm is shown in fig. 4 which can efficiently detect large blocks of different colors along with their orientation which will be used by robotic arm for grasping by aligning end-effector.

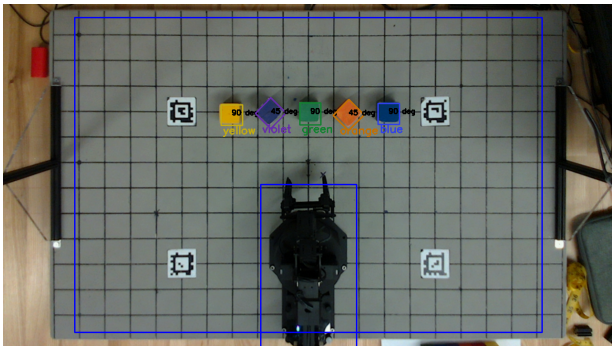

Fig. 4. Block Detector with Orientation.

E. Forward Kinematic Solution

Forward Kinematics (FK) involves utilizing a robot's kinematic equations to determine the position and orientation of the end effector in the world coordinate system, using specified joint parameter values. This process is analytical, with a closed-form solution and a unique result. The primary objective of FK is to ascertain the global frame positions of the joints based on the individual joint positions within the robot's frame [6].

In this project, we have employed the Denavit-Hartenberg parameters (DH parameters), as a standard set of parameters to describe the consecutive joint specific transformation matrices of the robotic manipulator. DH convention needs only 4 parameters by careful frmae choice. These parameters are:

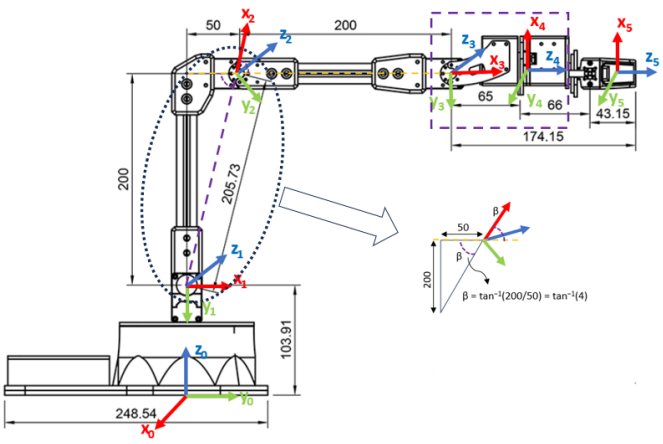

Fig. 5. RX200 arm schematic [7] with assigned frame annotations for DH parameters calculation.

- Joint Angle (θ): Represents the angle of rotation of the current joint about its z-axis to align the current (old) joint's x-axis with the new joint's x-axis.
- Joint Offset (d): Represents the offset along the previous joint's z-axis to reach the common normal (line aligned with the z-axes of the joint coordinate systems, used to establish a consistent frame of reference for defining the DH parameter). It describes the displacement from the previous joint's origin to the new joint's origin along the common z-axis.
- Link Length (a): Represents the displacement along the common x-axis from the old joint's origin to the new joint's origin. It describes how the joint moves along the x-axis to align the coordinate systems between the two joints.
- Link Twist (α): Represents angle about the the x-axis to align from the old joint's z-axis to the new joint's z-axis. It defines the joint axis orientation relative to the common normal.

As seen in the Fig. 5, all the arm joints are assigned frames systematically taking into account the axis conventions. Each joint i connects i-1th link to ith link. Based upon the robot axis directions we first assign the corresponding z-axis to all the joints. Subsequently, to assign x-axis to each ith joint we keep in mind that axis x_i is perpendicular to and intersects with axis z_{i-1}. For each joint we first observe rotation along the current z-axis (θ) for x-axis alignment; followed by translation along the current z-axis (d) i.e., in direction of common normal connecting the origins of current and subsequent joints along z-axis; translation along current x-axis (a); and finally rotation along current x-axis (α) for z-axis alignment. The sequence of which is shown in Eq.7, to create a joint specific transformation matrix, Eq.8 :

$$T_i = \text{Rot}_{z,\theta_i} \cdot \text{Trans}_{z,d_i} \cdot \text{Trans}_{x,a_i} \cdot \text{Rot}_{x,\alpha_i} \qquad (7)$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Furthermore, the complete FK equation for the robotic manipulator is assembled from individual joint homogeneous transformations, as shown below in Eq.9:

The complete FK equation for the robotic manipulator is assembled from each joint homogeneous transformations, shown in Eq.9:

$$\mathrm{T}_{overall} = \mathrm{T}_1(\mathrm{joint}_1) \cdot \cdots \cdot \mathrm{T}_n(\mathrm{joint}_n) = \begin{bmatrix} \mathrm{Rot}_n^0 & \mathrm{Loc}_n^0 \\ 0 & 1 \end{bmatrix} \tag{9}$$

Now once all the joint configuration parameters and coordinate frames are in place, as shown in Fig. 5, we fill up the DH parameter table (Table I) for our robotic manipulator using the joint specific convention sequence specified by Eq.7.

TABLE I
DH PARAMETERS FOR MANIPULATOR IN FIG. 5

Joint	θ_i (rad)	d_i (mm)	a_i (mm)	α_i (rad)
0	$\frac{\pi}{2}$	103.91	0	$-\frac{\pi}{2}$
1	$-\arctan(4)$	0	205.73	0
2	$\arctan(4)$	0	200	0
3	$-\frac{\pi}{2}$	0	0	$-\frac{\pi}{2}$
4	0	174.15	0	0

As seen in Fig. 5, in going from joint 1 to joint 2 of the robot arm we construct a unique shortest line segment from origin of z_1 to z_2 in order to determine the inclined direction of x_2 axis. To determine the exact inclination we use the trigonometric calculation shown in the sub-figure. Furthermore, we also consider the joint 4 to be at the end effector position by considering the total distance of 174.15 mm instead of mere 65 mm distance. This helps us to reduce rows in the DH table.

F. Inverse Kinematic Solution

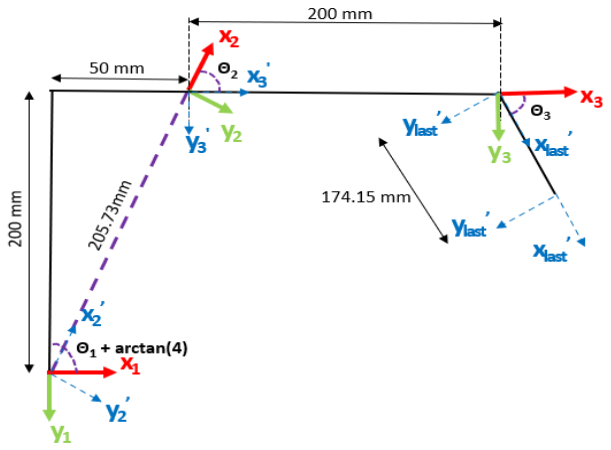

Fig. 6. RX200 arm shoulder to wrist joint skeletonized diagram.

Inverse kinematics (IK) is the mathematical process of calculating the variable joint parameters needed to place the end effector of robot manipulator in a given position and orientation (i.e., pose = (x,y,z,ϕ, θ, ψ)) relative to the start of the kinematic chain [8]. IK has closed form solution and non-unique solution exists [6]. Here, we observe IK as a root finding problem by solving a system of non-linear equations obtained from the FK transformation matrix. Following the RX200 arm schema as given in Fig. 5, we first calculate the base joint angle (θ_0) as, (refer Fig. 7 and Fig. 5)

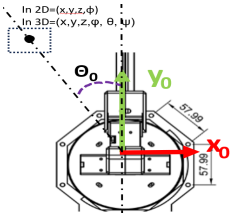

Fig. 7. θ_0 top view

$$\theta_0 = \arctan(-x, y) \tag{10}$$

In our case, smaller allied angle is taken, but $\theta_0 = \arctan(-x, y) + \pi$ is also a possible joint angle.

Subsequently, we compute the WristR joint angle, θ_4, which is used to align the gripper with block orientation (γ_{block}) during vertical grasp/release, such that

$$\theta_4 = \theta_0 - \gamma_{block}; where \gamma_{block} \in [0, \pi/2] \tag{11}$$

After, computing θ_0 and θ_4, we find the shoulder (θ_1), elbow (θ_2) and WristA (θ_3) joint angles simultaneously, by solving the 2 link RR manipulator IK sub-problem as illustrated in the Fig. 6. Here, we project the 3D frame coordinates of the manipulator to a 2D plane i.e., x-y coordinate frames. Then we compute the FK solution of the manipulator. The sequence of frame traversals to go from shoulder coordinate frame to wristA coordinate frame, based on Fig. 6 are as follows:

$$T = \mathrm{R}_{z,\theta_1} \cdot \mathrm{t}_{x,l1} \cdot \mathrm{R}_{z,\theta_2} \cdot \mathrm{t}_{x,l2} \cdot \mathrm{R}_{z,\theta_3} \cdot \mathrm{t}_{x,l3} \tag{12}$$

Here, R_{z,θ_i} and $\mathrm{t}_{x,li}$ signify the 2D rotation and translation about the corresponding current frame axis. The link lengths, l1, l2 & l3 are 205.73, 200 & 174.15 mm respectively. In Fig. 6 we can see that we traverse along the unique shortest line segment from shoulder to elbow joint. Hence,

$$\theta_1 = \theta_1 + \arctan(50/200) \tag{13}$$

After substituting all the manipulator data in the frame traversal equation (Eq.12) and multiplying all the homogeneous matrices, we get the overall transformation matrix (T) as follows:

$$T = \begin{bmatrix} \cos_{1-2-3} & \sin_{1-2-3} & f2 \\ \sin_{3-1+2} & \cos_{3+1-2} & f3 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

$$\begin{aligned} f2 &= (174.15 * \cos_{1-2-3}) + (200 * \cos_{1-2}) \\ &\quad + (205.73 * \cos_1) \\ f3 &= (174.15 * \sin_{2-1+3}) + (200 * \sin_{2-1}) \\ &\quad - (205.73 * \sin_1) \end{aligned}$$

Note in the above equations we have used a compressed form of writing the equation due to space constraints. For instance, "$\cos_{1-2-3}$" means $\cos(\theta_1 - \theta_2 - \theta_3)$.

Finally, along with f2,f3 we also solve for f1,

$$f1 = \theta_1 + \theta_2 + \theta_3$$

Now, by equating f1,f2 & f3 with zero,

$$f1 - \frac{\pi}{2} = 0 \tag{15}$$

$$103.91 + f2 - z = 0 \tag{16}$$

$$f3 - \sqrt{(x^2 + y^2)} = 0 \tag{17}$$

we formulate IK as a system of nonlinear equations in the "equ" (equation) function in "kinematics.py", which after setting base and wristR joint angles in "IK_geometric" function is being called to acquire the remaining joint angles. The numerical IK optimization problem is "equ" function is solved by using scipy's fsolve library.

In Eq.16 we add 103.91mm, the link length from base to shoulder, to f2 since the FK equations are with respect to Fig. 6, where the the kinematic chain starts from elbow which is 103.91mm above the base. Also, 3D frames are projected to 2D frames, the rotation components of frame change are taken care of by the x-y coordinates, but in reality the translation from elbow to end effector takes place in 3D i.e., perpendicular to the x-y rotation frames, hence we use z position and resultant vector of x-y positions to keep our optimization problem directionally constrained and accurate corresponding to the 3D world coordinates.

G. Path Planning

We employ the RX200 robotic manipulator for a variety of tasks, including Teach & Repeat, Click-to-grab/drop for safe manipulator operation.

1) Teach and Repeat: Teach and repeat is a widely used method for instructing robotic arms in performing repetitive tasks, offering a user-friendly approach without specialized expertise. This approach involves creating a system that accepts a series of distinct waypoints in joint space and directs the robotic arm to traverse the trajectory connecting these waypoints [6].

To initialize the "Teach" part of the task, the "Record Position" custom button is pressed to start recording the waypoints by appending them in a list in the "record" function in "state_machine.py". Along with recording the positions of manipulator using the RXArm class' "get_positions" method in the "record" function, the type of actions are also classified, i.e., when the gripper is open or closed (recorded using "Record Open Gripper" and "Record Close Gripper" custom buttons respectively) or recording. During the manual movement of the manipulator, to record the waypoints, the "Record Waypoint"/"Record Open Gripper" or "Record Close Gripper" buttons are pressed repeatedly at different positions, based on the type of tasks, in order to store the arm positions as waypoints.

Now, to initialize the "Repeat" part of the task, the pre-existing "Execute" button is pressed, to repeat the stored actions of the "Teach" cycle. The "execute" function found within the "state_machine.py" iterates through the recorded list of waypoints. It utilizes the RXArm class' "set_positions" method to guide the manipulator from its current joint configuration to a specified waypoint configuration. This movement is executed with a sleep time of 1 second. In scenarios where solely the initial and final waypoints are provided, we generate intermediary waypoints to ensure seamless and consistent trajectory, corresponding to the intended motion.

2) Click To Grab/Drop: To shift the blocks within the reachable workspace we implement the Click To Grab/Drop strategy within the "click_operation" function in "control_station.py". The algorithm is as follows:

- Initialization: Initialize the robot arm to its home position and enable camera calibration.
- Recording Mouse Click Positions: In the setup phase, the code records the current mouse position

(pt) with "mouse_eventpos()" in pixel coordinates. It then updates last_click array with the x and y coordinates of the click, preserving the click location. The "click_status" variable is also increased, to keep track of the number of clicks in the click-and-place process.

- Getting Depth Information: Retrieve the depth information z, in camera coordinates, at the clicked pixel position from the camera's raw depth frame.
- Converting Pixel Coordinates to World Coordinates: If the depth information is available (depth frame is not all zeros): We first convert pixel coordinates (pt) to homogeneous coordinates (pt_loc_old). If a birds-eye homography matrix is available, we apply it; otherwise, set current point coordinates (pt_loc) to recorded coordinates (pt_loc_old). Further, we calculate the z coordinate of pt_loc by retrieving corresponding depth value from the depth frame and find 3D world coordinates using camera matrices.
- Inverse Kinematics: Using the 3d world coordinates of click position, we will compute the end effector pose. We compute ϕ (by using the sum of inverse tangent of x over y world coordinate and block orientation from block detector) to pick the block in the current configuration; we keep θ and ψ as zero and π radians (for points beyond thresholded area of world frame and $\frac{\pi}{2}$ radians elsewhere) respectively. Next we calculate the joint angles by applying pose to inverse kinematics function ("IK_geometric") and use RXArm class' "set_positions" method to set the robot arm's positions to reach the calculated joint angles, ultimately reaching the desired click positions.
- Pick-and-Place: Based on the "click_status" we determine the operation mode. If the click status is odd, the arm performs the "pick" operation, wherein after adjusting the pose by lowering the end effector depth (z -= 200mm) and setting the desired height (z = 30mm), the "grasp()" method from RXArm class' "gripper()" function is used to grasp the block. Similarly, if the click status is even, the arm performs the "place" operation by first lowering the end effector to a desired height (z = 15mm), followed by using the same "gripper()" function's "release()" method. We also use an intermediate pose, where the base and elbow joint angles are set to $\frac{\pi}{2}$ radians in order to avoid collision with other blocks in the world frame while navigating to the "place" position after picking up the block.
- Repeat: Continue monitoring mouse click events and performing pick-and-place operations based on the user's input.

H. State Machine

- Pick 'n Sort: In this contest, we employ the block detection algorithm, as described in the earlier sections) to search for and arrange blocks in the workspace's positive-Y half plane, based upon their sizes. Till the viable workspace is empty of blocks, we continue the block pick-and-sort procedure. In order to identify block outlines and their areas, we take a photo of the workspace at the onset of each cycle and process it using our pre-defined block detection system. Based on a predetermined area threshold of 960-1050 sq. units, we categorize blocks as small or large ones. To properly position the end effector of the manipulator for grasping operation during pick cycle, we additionally use the orientation data, which also facilitates vertical and horizontal grasping maneuvers. Following block detection, we determine each block's centroid and harness this data to perform inverse kinematics in order to determine requisite joint angles. The robot picks up the block, acquires its temporary intermediary position before beginning the drop/release operation. Finally, large blocks go to the fourth quadrant, small ones to the third quadrant. The code incrementally updates the drop location to avoid drop area collision/inter-lap. The event repeats until there are zero blocks in the workspace image.
- Pick 'n Stack: In this contest, the block picking methodology is adopted from Pick 'n sort. We select the workspace's 3rd quadrant in the world coordinates, such that placement of blocks here is done keeping the stacks of large and small sized blocks disparate constituently from each other. To ensure seamless stacking, the drop release height is incremented after each drop and the dropping distance is decreased based on the block size.
- Line 'em Up: In this contest, we employed the color intensity and type identification capability of our block detector to decipher each block's color during the competition. Following color identification, we arranged the blocks according to the desired color chronology (ROYGBV) and then rapidly merge sorted the blocks arrangement. The sorted list produced was then compared to positions that were color-specifically pre-determined. By iterating over this list by leveraging the pre-planned location lists, we not only get 2 disparate size-sorted block lines in the 3rd workspace quadrant and, near-accurate block placement locations; but also can potentially include the missed workplace detections majority of the times.
- Stack 'em High: In this contest, we use an amalgamation of logic already developed in previous

contests. For instance, same as Line 'em Up contest, we leverage the color sorting scheme and sort all the detected blocks on the -ve half plane/3rd quadrant. Then we pick blocks by color and place them into 2 size-bifurcated stacks using the pick-and-drop methods devised in Pick 'n stack task. Then we go on updating the z coordinate of the end effector during the drop operation based on fixed incremental offsets, as mentioned and used in the Pick 'n stack task.

III. Results

A. Forward Kinematics

FK is implemented in "kinematics.py", wherein FK_dh function utilizes get_transform_from_dh function to return a combined transformation matrix (T) (Eq.9) representing the pose of the end effector, by using joint angles, DH parameters and no. of links as input. Now in order to get the pose of the end effector we then pass rotation part of T matrix through function get_euler_angles_from_T to get the ZYZ euler angles, which are then used as inputs along with the translation part of T matrix in the function get_pose_from_T to get the end effector pose vector.

The accuracy of the end effector pose in the workspace coordinates was demonstrated by comparing the outputs of the FK function's kinematic formulations with the known world positional estimates. The accuracy of our method was tested in Checkpoint 2 where we were asked to report the (x,y,z) position of the centre of the top of stack constituting of large blocks [6] placed at various known positions in the world coordinate frame in the workspace.

TABLE II
FK READINGS TAKEN AT THE CENTRE OF TOP OF STACK HAVE 6 LARGE BLOCKS

	(0,175)	(-300,-75)	(300,-75)	(300,325)
X mm	-0.24	-304.45	307.21	306.64
Y mm	173.69	-79.66	-77.38	326.74
Z mm	218.58	215.08	213.38	216.54
ϕ rad	-1.57	-1.52	-1.57	-1.56
θ rad	0.013	0.025	0.022	0.026
ψ rad	0	-0.689	-0.714	-0.744

In Table II we depict the proof of the tool-tip accuracy of our arm in FK mode. Considering, the most cumulative error prone case of stacking 6 blocks vertically at the known world coordinates and finding the tool-tip pose at the top of centre of stack of these large blocks, we can see that our readings in x-y coordinate frame have a very low average error rate of 2-3.5% (Table II) and the erro rate in height(z-xis) which considering the stacking of 6 large block has a value of 35 time 6 = 210 mm. Even along Z axis we can observe that our readings stay more or less within the same error range, hence testifying the robustness of our FK implementation. Considering, the ϕ, θ and ψ orientation angles, if we approach the stack such that the ground truth ϕ, θ and ψ are -1.57, 0.024 and -0.732 radians (from GUI)i.e.

B. Teach and Repeat

We used the Teach and Repeat methodology, as described in Section F-1 of this report, to teach the robot to cycle swapping blocks at locations (-100, 225) and (100, 225) through an intermediate location at (250,75). The results of joint angles vs. time can be seen in Fig.8.

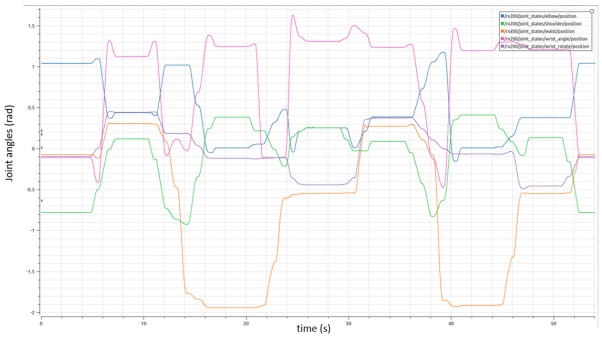

Fig. 8. RX200 arm joint angles configuration during teach and repeat.

We cycled 10 times through the Teach and repeat operation, then computed Fig.8 for the subsequent cycle. During the teach operation we used multiple incremental waypoints and had haults after pick and place operation in the block swapping cycle, hence you can see the extended constancy in joint angle plots.

C. Block Detection

As shown in fig. 4, our detection algorithm works accurately on the large size blocks. But as we have used area based block identification it is essential to test it on the small blocks which can seen in the fig. 9. From our evaluation we conclude that error margin in x and y axis is about $2-3\%$ and in z it is about $3-4\%$.

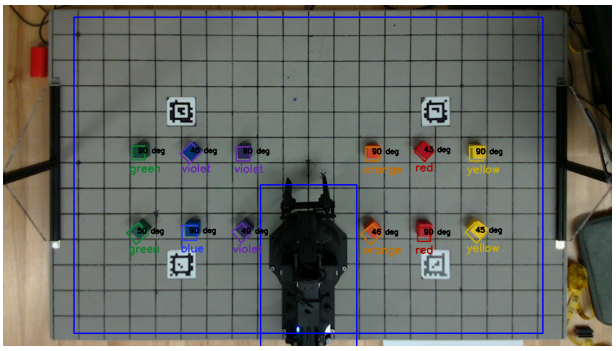

Fig. 9. Block Detection Algorithm on small blocks of different color

D. Contests

1) Pick 'n Sort: , our strategy was effectively able to identify the large and small blocks in world frame and was bale to sort it in third and Fourth quadrants. Although the approach was sound, we couldn't stay within the time limit, signaling a need for more efficient movements between waypoints. Even the gripping action were at a different offset signalling for more robust inverse kinematic solutions and faster movements.

2) Pick 'n Stack: , our algorithm was able to stack mainly 2-3 blocks out of the 6 blocks needed for assessment, but improper block detection at heights led to faulty gripping actions leading to unstable grips and collision as blocks kept on adding, resulting in our inability to complete the task within the time constraints. The unstacking solution if integrated properly could have solved the kinematic stability issues.

3) Line 'em Up: , our strategy successfully arranged the majority of the blocks in the correct order, with only minor issues encountered for specific blocks. We managed to address these issues by allowing the robot arm to extend to the end of its workspace in order to create a singularity solution which helped us avoid collision during lining tasks, enhancing the reliability of our approach. Due to time constraints and extended travel from singularity to block lining position there can be improvements in path planning.

4) Stack 'em High: , we faced similar stacking challenges as in Pick 'n Stack and the gripping errors as in Pick 'n Sort. Better IK evaluation with workspace specific configuration constraints can help us create more precise manipulation autonomy.

IV. Discussion

In our armlab implementation, we used auto calibration approach based on 4 April Tags located at known position on the board. First we performed the homography transform and then used pixel to point correspondence to fetch extrinsic matrix. One caveat with homography was scaling of the image since the image is of the form 1280px $\times$ 720px and the board is of the size 1000mm $\times$ 650mm. To overcome the scaling issue we used following approach:

$$height_{image} = height_{board}$$

From this we can conclude that 1mm $\approx$ 1.1px. One limiting factor of this approach is that size of board is required and automating it is something which could be looked into.

V. Conclusion

We presented the development of algorithms, strategies and implementation of how a 5-DoF robotic manipulator is capable of recognizing and interacting with variegated blocks using a RBG-D camera. The arithmetic foundations of our kinematic formulations and vision system have been thoroughly laid out with systematic implementation in this paper. Alongside that, solutions and improvement propositions have been made to address the issues in devising software of such robotic systems.

References

[1] AprilTag. April tag. [Online]. Available: https://april.eecs.umich.edu/software/apriltag

[2] OpenCV. Perspective-n-point pose computation. [Online]. Available: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html

[3] ——. Basic concepts of homography. [Online]. Available: https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html

[4] ——. Changing colorspaces. [Online]. Available: https://docs.opencv.org/4.x/df/d9d/tutorial_py_colorspaces.html

[5] ——. Contour features. [Online]. Available: https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html

[6] R. 550. Rob 550 armlab f23 document. [Online]. Available: https://drive.google.com/file/d/1du_QoUJ6VjDKu0NOGStlNBJXNBKX_qlZ/view?usp=sharing

[7] T. Robotics. Interbotix: Reactorx-200 desktop robot arm. [Online]. Available: https://www.trossenrobotics.com/reactorx-200-robot-arm.aspx.

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