Team 15 ROB 550 ArmLab Report

Nam Gyu Kil, https://www.overleaf.com/project/5fb532758096b635651d6be3 Tianyi Liu, Yingxue Wang

***Abstract*—In this lab, we programmed the ReactorX 200 (RX200) robotic arm to fulfill tasks of acting, sensing, and reasoning using Robot Operating System(ROS)[1]. We performed 5-DOF rigid-body transformations in homogeneous coordinate, implemented forward kinematics using 2 methods: Product of the Exponential (PoX)[2] and Denavit–Hartenberg (DH)[3] method, and solved inverse kinematics. Furthermore, we programmed an accurate block detection algorithm using OpenCV based on both depth and RGB information. In the end, we integrated the functionalities and designed workflows to complete the competition tasks that involve picking, placing, and stacking blocks.**

I. Introduction

ROBOTIC arms have been a fascination to many scientists for many years. Documents dating back to the 15th century show famous scientists, such as Leonardo Da Vinci, build one of the earliest programmable robot[4]. Since then, robotics arms have becoming increasingly advanced and have the ability to perform activities beyond mimicking human motion and manufacturing. For the ArmLab project, we looked into three aspects of robotics which are controls, sensing, and reasoning to enable the robotics arm to perform activities such as picking up a block and dropping it in the specified region. In this lab, we manipulated the RX200 Robot Arm which has 5 DOF manipulator and 7 Dynamixel servo motors with the ROS architecture.

II. Methodology

A. Smooth Trajectory Generation

The RX200 ships with driver libraries for commanding servo joints. By invoking $set\_joint\_positions()$ function, the user can specify a desired joint configuration, arm travel time and acceleration time. To ensure a smooth trajectory, the team implemented a smooth trajectory control algorithm. Each time $set\_position()$ is called, the algorithm iterates over each joint and calculates the maximum joint displacement. The arm travel time is set to be the maximum displacement divided by the max angular velocity. The acceleration time is set to be 1/4 of the total travel time. The angular velocity profile can be seen in figure 1. By applying this algorithm, the robot arm will drive to the desired configuration according to max angular velocity instead of a fixed time. This eliminates the possibility of sudden jerk when commanding joints to a configuration with large difference allowing for a much smoother trajectory of the overall arm motion.

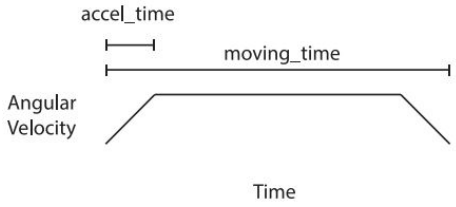


Fig. 1: Arm Angular velocity profile

B. Forward Kinematics

In this project, we implemented 2 different methods to calculate forward kinematics of the RX200 arm. The first method, PoX, constructs a series of rigid body transformations using skew vectors. The second method follows the DH convention by first assigning DH frames to find a series transformations using DH parameters. The detailed implementations are described in this section.

1) Convention: The robot manipulator consists of n joints (1 to n) and $n+1$ links (0 to n). By convention the base link (link 0) is stationary. For all other joints and links, When joint i is actuated, link i moves. Joint $i+1$ is fixed with respect to link i. Our RX 200 robot manipulator has $n = 5$ joints, and $n = 5$ links, from which Joint 1, 2, 3 belong to the arm body and joint 4, 5 belongs to the spherical wrist.

2) POX Method Explanation: The product of exponential is a very straight forward algorithm that can be used to find the location of the end-effector knowing the orientation of the joints. One of the first steps in the implementation of POX is to find the matrix M (home position) or the homogeneous transformation matrix that represents the end effector location when all the joint variables are 0. The M matrix is shown:

$$M = \begin{bmatrix} 1.00 & 0.00 & 0.00 & 423.00 \\ 0.00 & 1.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1.00 & 303.00 \\ 0.00 & 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Next the screw vectors for each of the joints needed to be calculated. The screw vector for the ith joint, S_i has the

following format $[\omega_x, \omega_y, \omega_z, v_x, v_y, v_z]^T$ where the first three terms are the normalized angular velocity about an axis of rotation and the last three terms are the linear velocity about the origin. The screw vectors are:

$$\mathrm{S}_1 = \begin{bmatrix}0.00 & 0.00 & 1.00 & 0.00 & 0.00 & 0.00\end{bmatrix}^T$$
$$\mathrm{S}_2 = \begin{bmatrix}0.00 & 1.00 & 0.00 & 0.00 & 0.00 & -104.00\end{bmatrix}^T$$
$$\mathrm{S}_3 = \begin{bmatrix}0.00 & 1.00 & 0.00 & 0.00 & 0.00 & -308.00\end{bmatrix}^T$$
$$\mathrm{S}_4 = \begin{bmatrix}0.00 & 1.00 & 0.00 & 0.00 & 0.00 & -394.00\end{bmatrix}^T$$
$$\mathrm{S}_5 = \begin{bmatrix}1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00\end{bmatrix}^T$$

For the forward kinematic calculation using POX, each screw vector must be represented in SE(3) as seen in Equation 2. Because SE(3) contains rotations, the angular component of S_i must also be written in SO(3) by representing as a skew symmetric matrix as seen in Equation 1.

$$[S] = \begin{bmatrix}[\omega] & v \\ 0 & 0\end{bmatrix} \in SE(3) \tag{1}$$

$$[w] = \begin{bmatrix}0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0\end{bmatrix} \in SO(3) \tag{2}$$

We then apply the Rodriques' Formula to find the matrix exponential based on the screw and rotation axis. The matrix exponential can be rewritten as seen in equation 3 which can be characterize by the homogeneous transformation matrix allowing us express the matrix exponential with the joint angle θ_i which is a known value.

$$e^{[S]\theta} = \begin{bmatrix}e^{[w]\theta} & (I\theta + (1 - cos(\theta)[\omega] + (\theta - sin(\theta)[\omega]^2)v \\ 0 & 1\end{bmatrix} \tag{3}$$

$$T(q) = e^{[S_1]\theta_1} e^{[S_2]\theta_2} e^{[S_3]\theta_3} e^{[S_4]\theta_4} e^{[S_5]\theta_5} M \tag{4}$$

Finally, we can then use the matrix exponential and the M matrix that we solved to find the final transformation that represents the orientation and the location end-effector position (Equation 3).

3) DH Table Method Explanation: There are two fundamental rules to follow when assigning reference frames in the DH convention: (1) x_i is perpendicular to z_{i-1} (2) x_i intersects z_{i-1}.

We assign the reference frames for each joint of the RX200 arm, and the frame assignment is shown in Appendix A. The procedure of assigning DH frames is as follows:

1) Assign $z_0, ..., z_{n-1}$ such that z_i is the axis of actuator of joint $i+1$.
2) Choose a base frame $O_0X_0Y_0Z_0$, X_0Y_0 any convenient manner.
3) Frame i is chosen based on frame $i-1$ for $i = 1, ..., n-1$. There are 3 cases:
   a) z_{i-1} and z_i are not coplanar, then there exists a unique shortest line segment from z_{i-1} to z_i perpendicular to both z_{i-1} and z_i. This defines x_i, the point where x_i intersects z_i is o_i.
   b) z_{i-1} and z_i intersect, x_i is the normal to the plane formed by z_{i-1} and z_i. o_i is the point of intersection of z_{i-1} and z_i
   c) z_{i-1} and z_i are parallel. Choose o_i anywhere along z_i and choose x_i to be any normal vector to z_{i-1} and z_i.
4) Choose $y_i, i = 0, ..., n-1$ to form right-handed orthogonal frames.
5) End effector frame: place o_n on the end effector and choose $x_n y_n z_n$ such that $x_n \perp z_{n-1}$, x_n intersects z_{n-1}.

By following the DH convention, we can define a 4-by-4 homogeneous frame transformation matrix H using a set of 4 parameters, representing the following values (Equation 5).

$$\begin{cases} a_i := |o_{i-1}o_i| \text{ along } x_i \\ d_i := |o_{i-1}o_i| \text{ along } z_{i-1} \\ \alpha := \angle z_{i-1}z_i \text{ about } x_i \\ \theta_i := \angle x_{i-1}x_i \text{ about } z_{i-1} \end{cases} \tag{5}$$

The actual DH Table of the RX200 arm used in this project is shown in Table I. The homogeneous matrix from frame $i-1$ to frame i, H_i^{i-1} is therefore be defined by substituting the DH parameters using Equation 6.

$$H_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The final homogeneous transformation is therefore

$$H_n^0 = H_1^0 H_2^1 ... H_n^{n-1}, \; H_n^0 = \begin{bmatrix} R_n^0 & O_n^0 \\ 0 & 1 \end{bmatrix} \tag{7}$$

from which we can extract the rotation R_n^0 and translation O_n^0 of the end-effector with respect to the base frame.

TABLE I: DH Table for RX200 Arm. All units are in mm.

link	a_i	α_i	d_i	θ_i
1	0	-$\pi/2$	103.91	θ_0
2	205.73	π	0	$\theta_1 - 1.3258$
3	200	0	0	$\theta_2 - 1.3258$
4	0	$\pi/2$	0	$\theta_3 - \pi/2$
5	0	0	131	0

C. Inverse Kinematics

The inverse kinematic is the process of mapping from robot end-effector world coordinates into joint configurations. With inverse kinematics (IK), the user can

command the robot arm with a desired workspace Cartesian coordinate, and the robot can calculate a feasible joint configuration to achieve the target. In this project, the team solved the IK problem using a geometrical approach. The RX200 arm has 5 joints [Figure 2]. For purpose of the competition, θ_5 is fixed to 0 deg. This section discusses the approach used to find θ_1 through θ_4. All IK related code are written in kinematics.py. The link variables used can be seen in Table II.

TABLE II: variables used in IK derivation. All units are in mm.

l_1	l_2	l_3	l_4
103.91	205.73	200.00	151.00

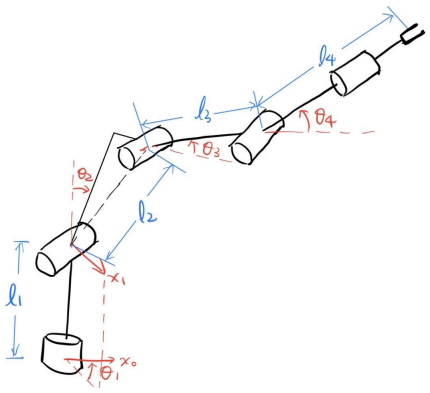


Fig. 2: Joint numbering in IK. Note l_2 is defined as the line connecting joint 2 and joint 3

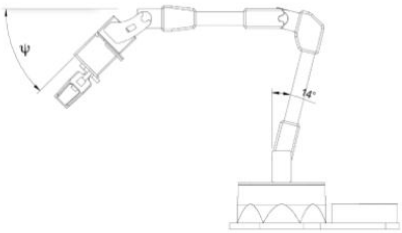


Fig. 3: ψ angle in IK. It is defined as the angle from the horizontal plane, or the rotation about the z-axis in frame 0.

For the RX200 arm, the IK function has four inputs: the x, y, z of target end-effector position, and the angle ψ of end-effector from the horizontal plane (Figure 3). The process begins by calculating θ_1. This part is done in code on line 255. The two solutions for θ_1 can be calculated using:

$$\theta_{11} = atan2(y, x) \tag{8}$$

$$\theta_{12} = \pi + atan2(y, x) \tag{9}$$

$$R = \begin{bmatrix} c_r c_p & -s_r c_y + c_r s_P s_y & s_r s_y + c_r s_p c_y \\ s_r c_p & c_r c_y + s_r s_p s_y & -c_r s_y + s_r s_p c_y \\ -s_p & c_p s_y & c_p c_y \end{bmatrix} \tag{10}$$

We can then write the pose of the end-effector as a homogeneous transformation matrix. This part is done on line 321. The translation part consists of the input x,y,z. To represent the end-effecor orientation in the world coordinate, the roll, pitch, yaw to rotation matrix equation (Equation 10) is used, in which roll equals θ_1, yaw equals ψ, and pitch is 0 for competition use case. To find the coordinates of joint 4 (referred to as O_c below), the follow equation is used, where O is the end-effector position:

$$Oc = O - l_4 R \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{11}$$

$$\theta_{31} = cos^{-1}\left(\frac{(r^2 + s^2) - l_2^2 - l_3^2}{2 l_2 l_3}\right) \tag{12}$$

$$\theta_{32} = -\theta_{31} \tag{13}$$

$$\theta_2 = atan2(s, r) - atan2(l_3 sin\theta_3, l_2 + l_3 cos\theta_3) \tag{14}$$

$$\theta_4 = -\theta_2 - \theta_3 - \psi \tag{15}$$

With O_c solved, finding θ_2 and θ_3 becomes a 2D RR arm problem [Figure 4]. This part is done on line 296. Using Equation 12, where r equals the x,y projection component of Oc and s equals the z component of Oc subtracted by l1, we can solve for the 2 solutions for θ_3. Using equation 14, we can solve for the according θ_2 with respect to 2 solutions of θ_3. Solving for θ_4 is straightforward, using equation 15. This part is done on line 300.

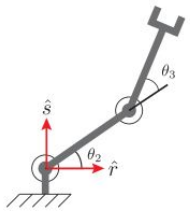


Fig. 4: Simplified drawing for θ_2 and θ_3

To conform to the factory settings of RX200, a few adjustments were needed. This part is done on line 305. In above IK calculations, joint 2 through 4 are assumed to be at 0 when link 2 to 4 parallel to world x,y plane. In reality, the RX200 defaults to configuration in [Figure 2]. Thus, a negative offset of 75.9 degree is required for θ_2. A positive offset of 75.9 is required for θ_3. Moreover, actual θ_2 rotates in the opposite direction compared to

above IK calculation assumption. A negative sign is applied to above θ_2 calculation in RX200 IK algorithm. A total of 4 theoretically feasible IK solutions are returned by the IK function.

D. Camera Calibration

In the camera calibration part, the team seeks to find a way to map a pixel on camera image back to its workspace coordinates [Figure 5]. This problem can be broken down into two sub problems. First, an object's 3D camera frame coordinates are projected onto camera's image plane. This matrix that achieves this projection is called intrinsic matrix [equation 16]. Second, there is also a homogeneous transformation that represents the world to camera coordinate transformation. This transformation is called the extrinsic matrix (equation 17).

The Intel Realsense 435i camera sensor used for this task consists of an RGB camera and an IR laser projector and an IR camera. The laser projector projects invisible IR lights onto the objects in the field of view, and then recorded by the IR camera which enables the sensor to measure the distance of objects. The camera comes from factory with pre-aligned depth and RGB image. Thus, no depth calibration function was required. This section introduces the methods used to find the intrinsic and extrinsic matrix for the lab camera setup.

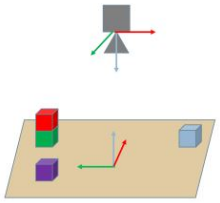


Fig. 5: Frames used in camera calibration problem. Red: x, Green: y, Blue: z.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z_c} M_{intrinsic} \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{16}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = M_{extrinsic} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{17}$$

The Realsense camera came from factory with its intrinsic matrix. The instrinsic matrix K (in pixels) can be retrieved through Realsense ROS topic:

$$K = \begin{bmatrix} 608.0704 & 0.0 & 317.8551 \\ 0.0 & 608.2894 & 239.5375 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

The extrinsic matrix was found with the help of Apriltags[5]. An Apriltag is placed on the arm's control unit with known coordinates in the workspace frame (Appendix B). A homogeneous transformation matrix from Apriltag frame to workspace frame can be written as Equation 18. The Apriltag ROS library provides tag pose detection in the camera frame. The detection ROS message contains position information in x,y,z, and orientation in quaternion. Using Equation 19, the quaternion representation can be converted into a rotation matrix. With a rotation matrix and x,y,z information, the homogeneous transformation matrix from camera frame to Apriltag frame can be written as Equation 20. With H_A^C and H_W^A solved, the extrinsic matrix can be represented as Equation 21.

$$H_W^A = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 141.6 \\ 0 & 0 & 1 & -37 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{18}$$

$$R = \begin{bmatrix} w^2+x^2-y^2-z^2 & 2xy-2wz & 2xz+2wy \\ 2xy+2wz & w^2-x^2+y^2-z^2 & 2yz-2wx \\ 2xz-2wy & 2yz+2wx & w^2-x^2-y^2+z^2 \end{bmatrix} \tag{19}$$

$$H_A^C = \begin{vmatrix} & & & x \\ & R & & y \\ & & & z \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{20}$$

$$H_W^C = H_A^C H_W^A \tag{21}$$

With the camera intrinsic and extrinsic matrix solved, we can tackle the workspace reconstruction problem. The process starts with knowing u,v coordinates in the 2D image plane. To convert this to the camera 3D frame, Equation 16 can be rewritten into Equation 22. In Equation 22, $Z(u,v)$ is the depth reading from camera at the pixel of interest. To convert from camera frame to workspace frame, Equation 17 can be rewritten as Equation 23. Note that $M_{extrinsic}$ is a homogeneous transformation the inverse of a homogeneous transformation needs to be done in two steps as Equation 24. The workspace reconstruction problem is therefore solved.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z(u,v) M_{intrinsic}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{22}$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} M_{extrinsic}^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{23}$$

$$H^{-1} = \begin{bmatrix} R^T & -R^T d \\ 0 & 1 \end{bmatrix} \tag{24}$$

E. Block Detection

In this project, we explored block detection methods with depth information from the IR camera as well as color image information from the RGB camera using OpenCV.
1) Detection in Depth Image: We use OpenCV function $mask = cv2.inRange(img, lower, upper)$ to create a binary mask of objects which has value of 1 when object height is above the operating surface. We then use function $\_, contours, hierarchy = cv2.findContours()$ to find a list of possible contours. The detection result from one image frame is shown in Figure 6. The depth image has little noise, therefore we do not need to deal with them specifically. However, we had to remove the contour representing the robot arm by threshold on the contour lengths, as the contour of the arm has significantly longer contours than the blocks.

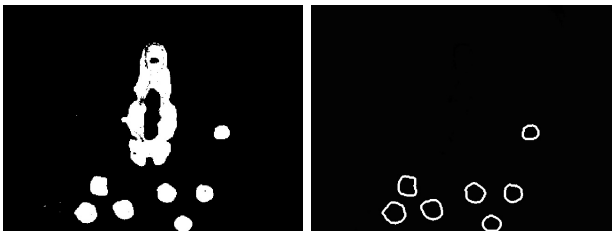
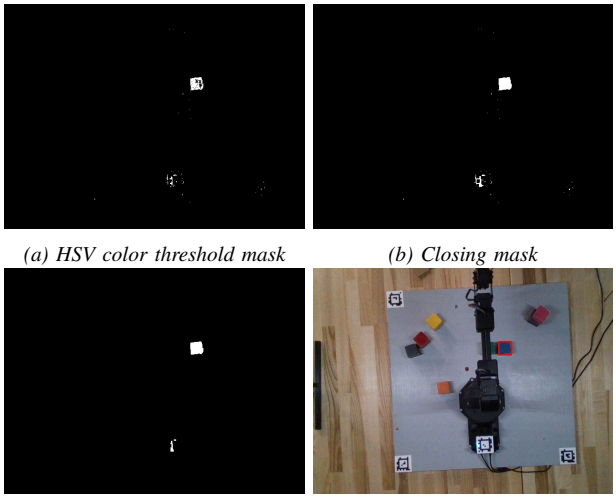


Fig. 6: Block detection in using depth information. Left: binary mask obtained from depth threshold, the small blob-like regions are mask of blocks and the large blob is the mask of the robot arm. Right: Contours of blocks after post-processing.

2) Detection in Color Image: One of the tasks is to be able to locate blocks of a specific color. We achieved this using the RGB image frames taken by the Realsense RGB Camera. The results are shown in Fig 7, and the steps of detecting color is listed as follows:

1) Extract one frame of RGB image from the camera stream and convert it into HSV format.
2) Threshold on the HSV values to obtain mask for different color. (Fig. 7a) The thresholds are found my running a HSV color-finding python script.
3) Post process the mask to remove noise and false positive detection using morphological filters:
   - Apply Closing filter to fill in the masked regions (Fig. 7b).
   - Apply Opening filter to remove noise. (Fig. 7c)
   - Apply Dilation filter to expand the masked region for a little so it fits the block contour even better.
4) Find contours of using the processed mask, generate bounding boxes using $rectangle = cv2.minAreaRect(contours)$. There could be 0, 1, or multiple bounding boxes found. Here is how we deal with these cases:
   - **0 Contour**: We see the detection found there is no block with the desired color.
   - **1 contour**: The contour is as desired.
   - > 1 **contours**: Calculate the area of the resulting bounding boxes. Choose the one has appropriate size as our final result because there is only one block for each color. The resulting contour is display in the control station.
5) Obtain the (x, y, z) value of the center of the block.



(a) HSV color threshold mask *(b) Closing mask*

(c) Opening mask *(d) Resulted bounding box (blue).*

Fig. 7: Block detection using RGB information of the Blue block.

III. Results

A. Smooth Trajectory Generation

The effect of smooth waypoint follower can be seen in [Figure 8], which plots the end-effector position when following a series of waypoints in [Table III]. With the smoothing algorithm, the magnitude of the slopes are reduced when tracking a waypoint with a large distance error. While when the waypoints are closer, the smooth algorithm has larger slopes. These improvements are more obvious when the RX200 is moving at high speed. In [Figure 9], the velocity profile has much fewer peaks when running the smooth algorithm.

TABLE III: Waypoints used in Figure 8. All units are in mm.

waypoint	θ_1	θ_2	θ_3	θ_4	θ_5
1	-$\pi/4$	-0.5	0	$\pi/2$	0
2	0	0	0	0	0
3	$\pi/8$	-0.5	0	0	$\pi/2$
4	$\pi/4$	0.5	0	$\pi/2$	0
5	0	0	0	0	0

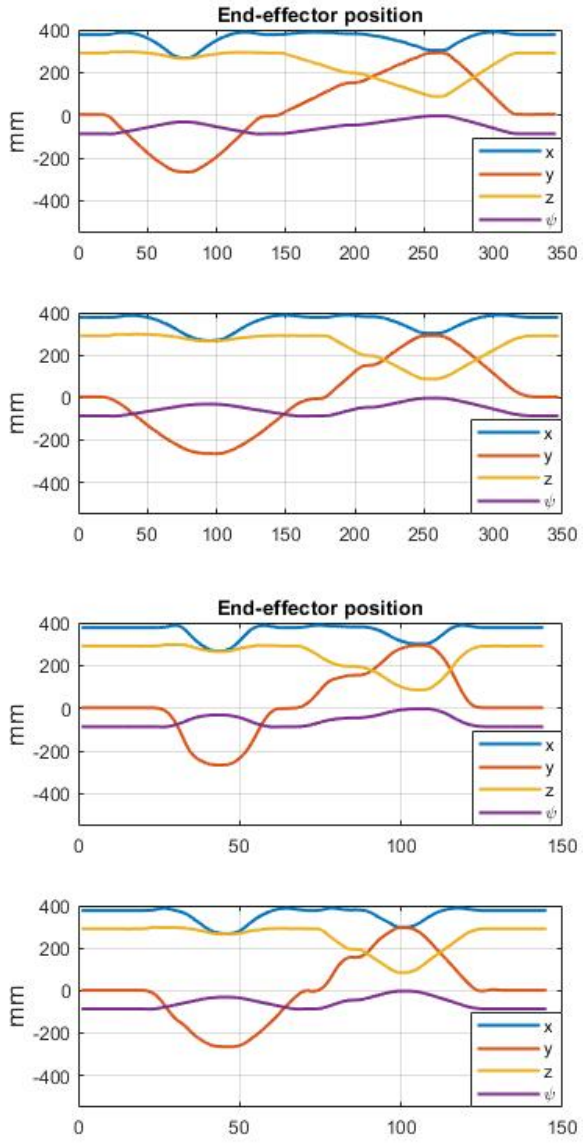


Fig. 8: End-effector position without (top) and with (second) smooth algorithm at low speed. Third (without) and fourth (with) at high speed

B. Forward Kinematics

The team compared the results from both methods. To evaluate the accuracy of the forward kinematics, we chose three poses: base 0 degrees, base -45 degrees, and base 45 degrees (Figure 10). Our reasoning behind these poses was because less noise would be introduced from the servo motors if we based our evaluation on only one of the servo motors, the base, moving. The ground-truth pose of the end-effector [x, y, z, θ] for the three different poses are the following:

- Base 0 deg: [423.5 mm, 0.0 mm, 303.91 mm]
- Base -45 deg: [299.21 mm, -299.21 mm, 303.91 mm]

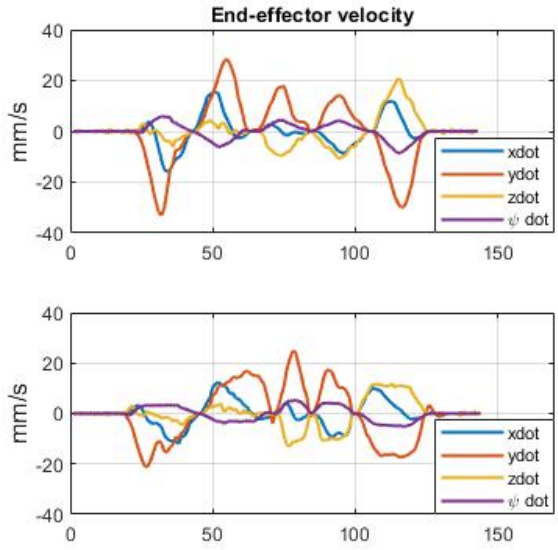


Fig. 9: End-effector velocity without (top) and with (second) smooth algorithm at high speed

mm]

- Base 45 deg: [299.21 mm, 299.21 mm, 303.91 mm]

Table IV and table V shows the average error (based on 20 recordings) in the end-effector positions for the three different poses. Much of the error seems to be introduced from the noise from the servo-motors. This was more apparent from base 45 degree and base -45 degree poses as moving the base servo from the home position had higher error than at base 0 degree position. Additionally, the values of the end-effector position would oscillate back and forth and though the servo motors were incredibly accurate, they still introduced significant noise. Furthermore, there seems to be some offset introduced every time the robot arm was initialized. The high error value of the DH seems to be from an offset of the x, y, z, and ϕ. Aside from these observations, our team did not see a significant difference in result between the POX and DH performance.

TABLE IV: DH RMS error for x,y, z in mm and ϕ in deg

Pose	$Base-0deg$	$Base-45deg$	$Base45deg$
x	3.2	4.1	3.9
y	2.3	2.5	3.1
z	2.0	2.2	1.9
ϕ	1.3	3.2	3.9

TABLE V: POX RMS error for x,y, z in mm and ϕ in deg

Pose	$Base-0deg$	$Base-45deg$	$Base45deg$
x	1.2	2.1	1.7
y	1.4	1.9	2.2
z	1.0	1.2	1.1
ϕ	0.1	0.2	0.2

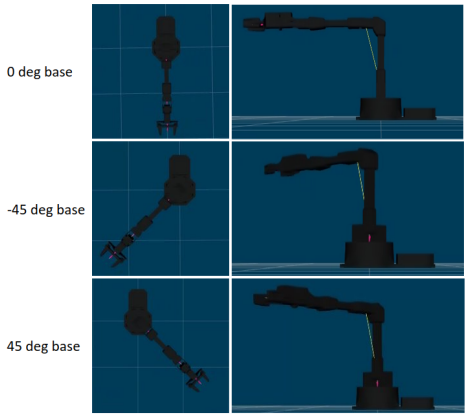


Fig. 10: Three different poses with different base angles for evaluation of forward kinematics

Figure 10 and Table IV and Table V shows the comparison between the POX and DH methods. As it can be see the POX/DH has a smaller error and also more consistent in its results. This is most likely due to numerical error caused by cosine and sine operations on small angles as this two methods should be equivalent.
For the competition, our group decided to implement the DH for the forward kinematics as it had lower Root-Mean-Square Errors (RMSE).

C. Camera Calibration

The camera calibration was verified using the three Apriltags on corners of the lab station board (Appendix C). The verification was conducted on Station 1. The ground truth dimensions were provided by the ROB 550 staff team. The calibrated dimension results were obtained using the control station. The worksapce z coordiantes of the corner Apriltags were not given. Thus, the team assumes a perfectly flat table for the ground truth z, although the table the station rests on may have an unknown tilt. In initial tests, the team discovered large fluctuations in workspace z coordinate in calibration results. The cause of this error could be from uncertainties in Apriltag orientation detection. An error analysis between ground truth and calibration results can be seen in (Table VI). It can be seen from (Table VI) that the calibration has a satisfying result, with error in each dimension not exceeding 5 mm.

TABLE VI: Tag ground truth workspace coordinates VS calibration results. All units are in mm. Subscript t indicates ground truth. Subscript c indicates calibration.

tag	x_t	x_c	y_t	y_c	z_t	z_c
2	-207.5	-204.7	277.5	281.9	0.0	0.4
3	347.5	354.2	277.5	271.4	0.0	2.7
4	-207.5	-204.8	-277.5	-274.4	0.0	3.1

D. Block Detection

To verify the accuracy of the block detector, we overlap the founded bounding box on top of the image frame and display it from the control station. For example, Fig.7d shows a bounding box for the blue block drawn in red. By comparing the positions and orientations of the bounding boxes with the actual blocks's, we concluded that the detection algorithm is accurate with an acceptable degree of errors.
To verify this numerically, we placed the blocks every 10 cm across the board in the simulation environment by modifying the gazebo setup. At each location, we compared the detection-returned position (x, y) with its ground truth values, and calculate its Euclidean error (square root error). The result is shown in Fig.11, where the heatmap itself represents the board for the operation. We did not do comparison on the orientation because we did not consider it when perform pick/drop actions. Here, we assumed that the blocks are all placed directly on the board with no stacking. Since the errors are all relatively small comparing to the size of a block, we considered our detection algorithm to be accurate.

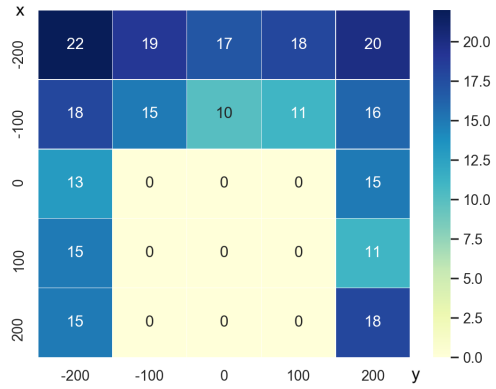


Fig. 11: Heat map for Euclidean errors of blocks on different board location. All units in mm. The lower center values are 0s because the locations are too close to the arm base thus being ignored.

E. Competition

1) Event 1. Pick'n Stack: The goal of this competition is to move 3 blocks of color {Red, Green, Blue } from the right half of the board and stack them together at any location on the other side of the board. We dissembled the task into 3 steps: detect, move, and stack.
Since we can stack the block on an arbitrary location on the left-half plane, we simply obtain the desired stacking location P_d in the world frame by negating the block's y-coordinate so that it is symmetric about the

center line. To stack the following blocks, we re-detect the previous block and use its current location as P_d, and then increase its z-coordinate by the height of one block. By experiment, we found that the this approach tolerates the error occurred during the operation well. The arm is commanded to move by receiving a desired world location and solve an Inverse Kinematic problem to obtain the set of desired joint angles. Each time we want to pick or place an object, we have the robot arm to arrive at a location above the desired location first before reaching the target as a simple way to avoid collisions with other surrounding objects.
The pesudo-code for the workflow is shown as follows in Algorithm 1.

Algorithm 1: Workflow for Pick'n Stack task

```
Begin
for color in {Red, Green, Blue} do
    block_location = block_detection(color);
    if first block to be stack:
     block_location.y = - block_location.y;
     move_arm(block_location);
    else:
     prev_block_location =
       block_detection(prev_color)
     move_arm(prev_block_location);
end
End ∅
```

Performance Evaluation: The robot arm was able to successfully finish the task. It detects the desired color block with minimal error, pick up the blocks and move it to desire location smoothly. The arm sometimes fails to pick up a block due to noise resulted from forward/inverse kinematics calculation and block orientation, since we set the angle of joint 5 to be always zero and we always approach the target from a 30-degree angle from the vertical line as a simplification of the task.
2) Event 2. Line 'em Up: The color block detection logic follows Event 1. However, since the blocks can be stacked, we first need to un-stack them, as one desired color may be block by some other blocks on the top. Therefore we need to use the height information obtained from the IR camera to find the stacks and un-stack them. Afterwards, we proceed like Event 1 and place the object in the desired locations.
3) Event 3. Line 'em High: Though similar to Event 1 and 2, in this task we are trying to stack the blocks as tall as possible, therefore we need to consider the failure cases and do not let one stacking failure affect the following ones. Therefore, combination of the color information and height information are both useful to achieve this goal.
4) Summary: For the competition events, we were able to successfully attempt Event 1. The robot has fully functionalities to compete in Event 2 and 3 and bonus drawing with assisted teaching. The team chose to not compete in autonomous drawing. Possible improvements will be discussed in the Discussion section.

IV. Discussion

There are some detailed and more sophisticated design that we could adopt to improve the arm's performance of the competition tasks.
One possible improvement is to design an obstacle avoidance algorithm with finer trajectory for the end effector to follow. Our current design first have the arm arrives at a location above the target and then approach downwards. However, this design can sometimes be too coarse that it knocks down other objects as it blindly sweeps around the work-space. A fix for this issue will improve the performance of the "Stack 'em high" task.
When solving the inverse kinematic problem, our current design choice is to always choosing the elbow-up posture of the arm. This may introduce unnecessary and abrupt changes when the robot arm is trying to arrive at some specific configurations. It is beneficial to have a set of carefully-designed rules to choose between elbow-down/elbow-up posture of the arm.
Moreover, instead of using a simple mask based on color and filtering techniques to find the location of the blocks, other algorithms, such as Ford-Fulkerson[6] or deep learning networks such as yolo[7] could have been used. Using Ford-Fulkerson, a segmentation of background and foreground would given results that are more robust to lighting and color variation in the camera image. Tuning the graph for color similarity and proximity would have allowed for an efficient algorithm to detect the blocks and use a simple threshold based on colors to find certain color blocks. Additionally, if given more computational resources, CNN such as yolo could have been implemented to find the bounding boxes of the coloreed blocks. Additional layers could have been added to the network to output the position of the blocks. Both these methods would have resulted in a much more accurate detection of the colored blocks.

References

[1] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org
[2] R. W. Brockett, "Robotic manipulators and the product of exponentials formula," in *Mathematical theory of networks and systems*. Springer, 1984, pp. 120–129.
[3] P. I. Corke, "A simple and systematic approach to assigning denavit–hartenberg parameters," *IEEE transactions on robotics*, vol. 23, no. 3, pp. 590–594, 2007.
[4] M. E. Moran, "Evolution of robotic arms," *Journal of robotic surgery*, vol. 1, no. 2, pp. 103–111, 2007.
[5] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3400–3407.
[6] V. Vatter, "Graphs, flows and the ford-fulkerson algorithm," *Tersedia di: http://www. math. ufl. edu/˜ vatter/teaching/flow. pdf.[diakses 10 maret 2013]*, 2004.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[8] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2006. [Online]. Available: http://www.probabilistic-robotics.org/
[9] I. S. D. Siegwart, R. Nourbakhsh, *Introdution to Autonomous Mobile Robots*. The MIT Press, 2011.
[10] S. Kalan, S. Chauhan, R. F. Coelho, M. A. Orvieto, I. R. Camacho, K. J. Palmer, and V. R. Patel, "History of robotic surgery," *Journal of Robotic Surgery*, vol. 4, no. 3, pp. 141–147, 2010.

V. Appendix

A. DH Frame Assignment for RX200 Robot Arm

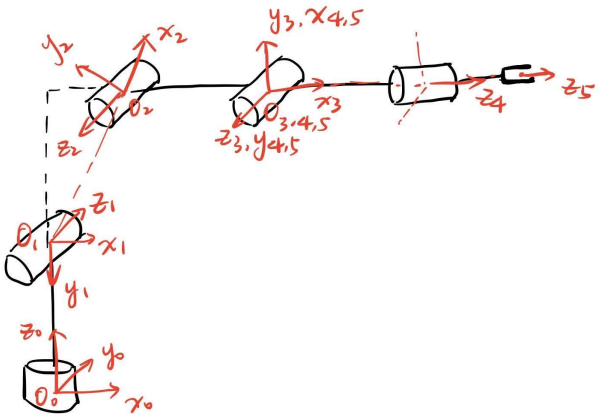


Fig. 12: DH frame assignment for RX200 Robot Arm.

B. Apriltag Position in Workspace Frame

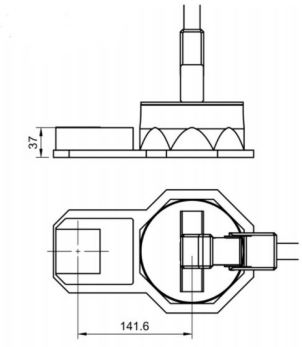


Fig. 13: Apriltag position in workspace frame. All units in mm

C. Lab Station Board Layout and Apriltag Location

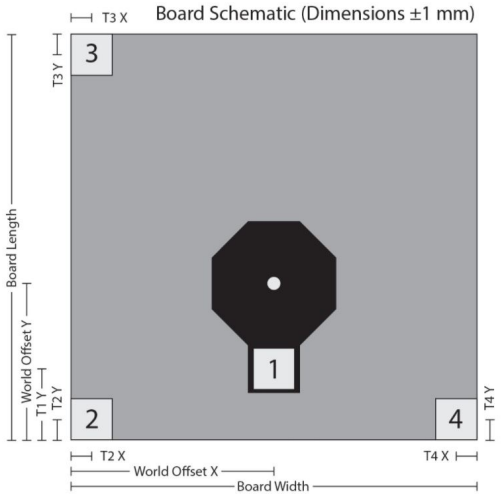


Fig. 14: Lab station board layout and Apriltag location