ArmLab Report

Yi-Cheng Liu, Frank Yifeng Lai, Shrey Shah
{liuyiche, fylai, shreyzz}@umich.edu

***Abstract*—In this lab, the aim is to enhance RX200 arm's autonomy by integrating computer vision, kinematics, and path planning to the autonomous manipulation of blocks in the workspace. The report encompasses the development and application of forward and inverse kinematics for accurate modeling and control of the manipulator. On the sensing front, OpenCV is utilized for object detection, and depth sensors are used to determine their accurate locations. Furthermore, the work involves formulating algorithms for path planning and facilitating autonomous decision-making by the robot.**

I. Introduction

Robotic arm manipulation presents a complex yet vital subject matter within the realm of advanced technology. Effective control of a robotic arm requires demonstrating an understanding of kinematics, computer vision, and control theory. This project explores the autonomous manipulation of blocks using a 6DOF robotic arm. The project concentrates on implementing forward and inverse kinematics to accurately control the robotic through the precise control of joint angles. To autonomously manipulate blocks, the system performs block detection using a separate camera and relays useful data to the arm. This process uses computer vision to detect blocks and coordinate transformation to locate the blocks in homogeneous coordinates. The following report details the process of programming a robot to autonomously manipulate blocks. It also assesses the robot's programming by presenting simple tasks. The report finally discusses these findings and reflects on the limitations and challenges of robotic arms.

II. Computer Vision

Computer vision is a core functionality of the robotic system. The goal of the computer vision system is to detect the location of blocks and items of interest. The ability to detect blocks allows the arm system to achieve autonomy in completing defined tasks. The computer vision is driven by a RealSense depth camera mounted to the top of the work area.

To make use of the vision system, a conversion between the pixel coordinates and real-world coordinates needs to be performed to allow the arm to understand graphical inputs and detections. Furthermore, a robust block detector must be implemented to allow the robot to autonomously detect blocks and correctly manipulate them. This section details the process of creating an algorithm that can detect blocks and express their location in world frame coordinates.

A. Camera Intrinsic Calibration

The optical and the mechanical properties of the camera create some errors in the form of distortion. These errors make it difficult to map the object detections from the camera to the workspace. The camera distortion is due to the mechanics of the camera and is often difficult to calibrate.

A ROS node is provided in this lab that allows for calibration of the camera distortion values by moving a checkerboard tool. The node compares the locations, size, and skew of the checkerboard pattern against the expected values to determine the intrinsic matrix that converts pixel values to real-world coordinates in the camera frame. A picture of this process is shown in Fig.10 located in the appendix. This process is repeated 5 times from which an intrinsic matrix for image-to-camera frame conversion can be derived. The results of the intrinsic calibration are shown in Table I.

Method	Intrinsic Matrix
Factory	$\begin{bmatrix} 900.715 & 0 & 652.287 \\ 0 & 900.192 & 358.36 \\ 0 & 0 & 1 \end{bmatrix}$
Measured	$\begin{bmatrix} 911.624 & 0 & 647.808 \\ 0 & 912.855 & 353.988 \\ 0 & 0 & 1 \end{bmatrix}$

TABLE I: Comparison between Factory and Measured Intrinsic Matrices

Multiplying the depth information in the camera frame Z_c and the intrinsic matrix to a pixel coordinate results in the 3D coordinates of a point concerning the location of the camera.

It is noteworthy that the cameras broadcast a factory intrinsic matrix that is a good estimate of the intrinsic characteristics of the camera. This matrix is provided in table I. The measured matrix and the factory settings are highly similar to each other. Sources of error for both matrices could come from machining differences, algorithm rounding errors, and computer vision

detection errors. These sources of error, are not large in experimentation and correspond to 2-3mm of error. Both matrices were tested on the system using equation 4 in the process further explored in section II-C. In experimentation, however, the measured intrinsic matrix was more accurate in matching real-world coordinates with the corresponding pixel values the rest of the report will assume the use of the measured intrinsic matrix in table I.

B. Camera Extrinsic Calibration

The process of determining the extrinsic calibration of the camera H, is the process of determining the translation and rotation of the camera with respect to the base frame. That is, determining the R and T matrices in equation 1. Where the rotation is $R_{xyz} \in SO(3)$ and T is the translation in Euclidean coordinates. A naive way to measure this is to simply measure out the positional offsets in the workspace using a tape measure and assume a simple π degree rotation in the x-axis such that the z-axis points out of the workspace. This method of taking simple measurements to construct the homogeneous transformation matrix was experimented with first but produced inconsistent results. The main source of error is that there is non-negligible rotation in all axes caused by errors in camera mounting. This error is difficult to measure and causes significant deviation in position conversion, especially at the edges of the frame. This matrix is also not reliable because it takes a significant amount of effort to calibrate. This causes issues if the pose of the camera changes each time due to small variations in the workspace. It is not reliable to use this matrix every time. The final best guess measured extrinsic matrix is shown in table II.

$$H = \begin{bmatrix} R_{xyz} & T \\ 0^T & 1 \end{bmatrix} \tag{1}$$

To combat this issue of unknown rotation offsets and accurately measure the rotation of the camera frame, a solution is to use an automatic calibration method using points whose locations are known in 3D coordinates. To determine the pixel locations of known 3D positions, the use of Apriltags is employed. Apriltags are unique patterns that are easy to detect by a computer. The project includes a detector for these tags and can reliably detect the center of a tag in the image frame. The Apriltag detections are shown in image 3. These detections are very accurate and reliable.

A perspective-n-point (PnP) pose computation algorithm is used to automatically compute the extrinsic matrix of the camera. This method takes points in the real-world plane and projects them onto the image frame using the perspective projection model and the camera intrinsic matrix. 8 Apriltags are used and placed in the workspace during the calibration process to create an 8-point projection. Figure 1 shows the location of all the Apriltags in the workspace when calibrating. The extrinsic matrix can be computed by solving for the matrix given real-world Apriltag centroid locations and the detected Apriltag centroid locations.

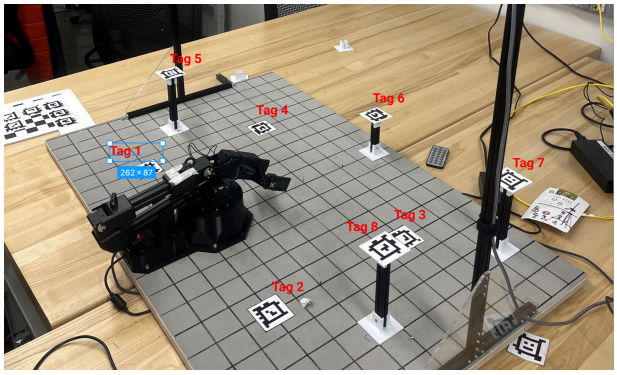

Fig. 1: Calibration Setup

Method	Extrinsic Matrix
SolvePnP	$\begin{bmatrix} 0.999 & -0.039 & -0.015 & 12.557 \\ -0.037 & -0.988 & 0.15 & 191.363 \\ -0.021 & -0.149 & -0.989 & 1036.06 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Measured	$\begin{bmatrix} 0.999 & -0.034 & 0.009 & 12 \\ -0.035 & -0.965 & 0.259 & 172 \\ 0 & -0.259 & -0.966 & 1045 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

TABLE II: Comparison between SolvePnP and Measured Extrinsic Matrices

This method is robust in that it is able to compensate for changes in camera position without needing to take measurements again. Although the two matricies are similar, the accuracy is also higher compared to the measured matrix. Figure 2 shows the error heat map for both the coordinate transform using PnP and the measured values. The brighter values in the middle represent the arm. Although both transformations produce higher errors near the edges of the workspace, the transformations using PnP are comparatively lower as shown by the dimmer heat map. This comparison makes it evident that although error is high and non-negligible for both methods, PnP provides a slightly more accurate picture. The rest of this report assumes the use of the PnP matrix. This matrix serves as a conversion from the world frame P_w to the camera frame P_c denoted as E.

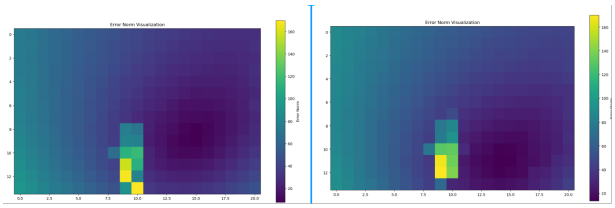

Fig. 2: Comparison of extrinsic matrices, left image shows the perspective-n-point solution, and right graph shows the measured solution

C. Coordinate Transformation

The transformation complete transformation requires two main processes. First is a camera intrinsic transformation to remove camera distortion and express points of interest from pixel coordinates to real-world 2-dimensional coordinates from the camera's perspective. Pixel coordinates will be denoted as P_p in equation 3. This transformation is explained in section II-A using the measured K matrix shown in equation I. The conversion multiplied by the depth information from the depth sensor gives accurate 3D coordinates with respect to the camera P_c.

In addition to the above 2 steps, a perspective transformation is included to warp and straighten the image presented in the control station. The function used for this process is $cv.findHomography()$ supplied with the Apriltags with IDs 1 through 4 as shown in image 1. This function is a feature-matching algorithm involving an $SE2 \rightarrow SE2$ homogeneous transformation matrix. The energy in the matrix represents a 2D rotation and translation that shifts and rotates an image so that the supplied points on the original image match the given points.

Figure 3 shows the effect of the perspective transformation. Grid points shown in figure 3 are projected onto the transformed image. The accuracy of the transformation is qualitatively assessed by verifying that the projected grid points match up well with the actual grid points in the image. The homography matrix H generated is shown in 2

$$H = \begin{bmatrix} 1.050 & -0.128 & -38.050 \\ 0.038 & 1.006 & -66.602 \\ 4.99 \times 10^{-6} & -1.4 \times 10^{-4} & 1 \end{bmatrix} \quad (2)$$

These three steps constitute a transformation that converts pixel coordinates in a perspective-transformed image P'_p to coordinates to 3-dimensional world coordinates with the robot's base as the origin P_w. The full transformation is shown in equation 4. Using this pipeline, we are able to convert image coordinates to real-world poses locations mouse-click events and block detections.

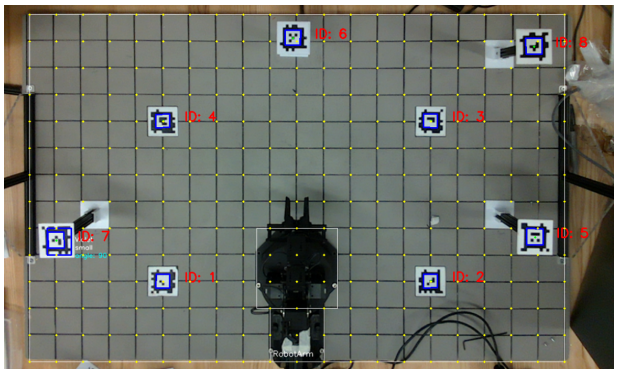

Fig. 3: Projected gridpoints after homography

$$P_w = \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}, P'_p = \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} \quad (3)$$

$$P_w = E^{-1} Z_c K^{-1} H^{-1} P'_p \quad (4)$$

D. Block Detection

To operate on the blocks, the pose and attributes of every block need to be detected. The provided blocks come with a set of characteristics, including color attributes, and size categories. There are a total of six unique colors, including blue, green, yellow, orange, violet, and red. Blocks are classified into three distinct size categories based on their area in the captured image: small, large, and distracting objects. This classification allows the system to sort and identify blocks based on their sizes while actively disregarding irrelevant objects that don't fit within the constraints. A general procedure for detecting blocks is as follows:

*1) **Define Region of Interest (ROI)**:* An ROI is set to focus only on a certain region of the image. We set the main ROI(the board) and also exclude certain regions within that main ROI (like the robotic arm). When the blocks have to be detected near the robot arm, we separate the undesired robot arm into two sections that fit the robot arm better.

*2) **Color Classification through Color Space**:* To mask the given image and get the blocks, we have to know the ranges of different colors in its individual color spaces. We use two color spaces, HSV (Hue, Saturation, Value) and YCrCb (Luminance and Chrominance). We designed a GUI utility (Fig.12) to find the color ranges. By supplying an image, we can adjust the sliders to fine-tune the lower and upper channel boundaries of the selected color space, which is crucial for precise color segmentation.

The $processColorMask$ function adeptly identifies block attributes like center(warped pixel coordinate),

size(isLarge), and orientation in real time with remarkable precision. By leveraging predefined color ranges, previously acquired by the GUI utility, it utilizes the $cv2.inRange()$ method to produce tailored color masks for each block color. This results in a binary mask where pixels within the designated color spectrum appear white, signifying potential blocks, while the rest remain black. However, raw color masks often contain noise or minor unwanted artifacts. We address this using morphological operations. An initial "opening" operation, comprising erosion followed by dilation, removes small noise. This is succeeded by a "closing" operation, using dilation and then erosion, to fill minor gaps in the detected blocks. This process helps filter out false positives in the detection.

Post noise reduction, we then extract the contours from the refined masks. Using the $cv2.findContours()$ function, the contours, essentially the boundaries of potential block areas, are extracted. Since the detected contours could be too small or large, we set a specific range of threshold to limit the detection. Each contour is then scrutinized to determine if it corresponds to a block. Here, the $cv2.minAreaRect$ function proves instrumental by calculating the minimum enclosing rectangle for each contour, offering information such as the centroid(warped pixel coordinate), dimensions, and rotation angle(Fig). In addition to evaluating the contour's area to verify it falls within the constraints of small and large blocks, disproportionate dimensions are also checked. If a detected block's height significantly outweighs its width, or vice versa, it's labeled as an irregular detection, including arch, rectangular, and triangular blocks. Otherwise, it's classified based on its unique color, size, and dimension.

3) ***Visualization****:* To verify the accuracy of color and size identification for each block in the warped station image, we introduce the function $addBlockInfo(img)$. This function annotates each block with relevant information sourced from $block\_detections$, where we've pre-organized the centroid, bounding box dimensions, color, size, and rotation angle for every block. Initially, some contours may flicker as the masked image stability is yet to be optimized. The problem is solved after tuning the color spaces. Using this method, the block detector achieves high accuracy verified qualitatively as shown in figure 4.

III. Robot Control

Before using forward kinematics and inverse kinematics, implement teach and repeat to verify the robot's ability to report and move to joint angles. We use the $get\_positions$ function to record the current joint angles and append them to the taught waypoints. The taught waypoints have an additional entry which includes

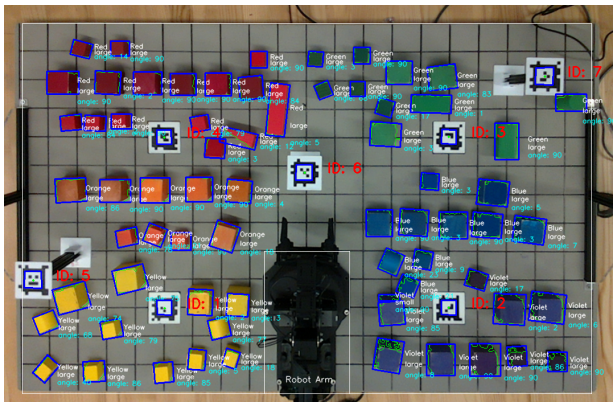

Fig. 4: Visualization of the block detector

gripper status (grab and drop). In a block-swapping task, the robot is able to repeat the motions up to 10 times. Fig. 5 shows every angle in one iteration of swapping the blocks. Fig. 6 shows that the end effector position which looks like it switches between three locations.

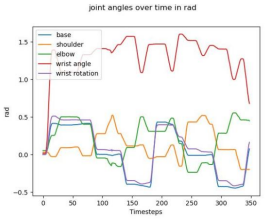

Fig. 5: Joint angles

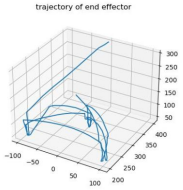

Fig. 6: End effector pose

A. Forward Kinematic Solution

Forward kinematics (FK) plays a pivotal role in robotic manipulation. Its primary objective is to determine the position of the joints in the global frame, given their positions in the robot's local frame. This process predicts the end effector's location, which is the operational tip of a robot, in a world coordinate. By relying on the theoretical forward kinematics solution, we can get a relation between the robot and the real world eliminating any errors or offsets as needed.

There are multiple methodologies to approach the implementation of FK. Among these, the Denavit-Hartenberg (DH) table and the Product of Exponentials (PoX) method stand out.

1) *Denavit-Hartenberg (DH) Method*
   This systematic approach defines joint parameters to determine the end effector's position.
2) *Product of Exponentials (PoX)*
   Utilizing screw vectors and an M-matrix, this technique computes the end effector's position, offering a more generalized formulation.

Our station employed the analytical approach where we decouple the 5 rotational joints to a 1R-2R-2R problem Let O_5 be the end effector frame. Provided the rotation matrix R_5^0 and the final link distance, we can move back to the O_4 frame using the equation given below. From Fig. **??**, it is given that O_3 and O_4 coincide which means the frame O_3's location is not dependent on θ_4. Fig. 9 shows the decoupled first three joint angles. (x,y,z) are obtained from the frame O_4.

Two distinct methodologies were considered for frame assignment. To avoid singularity configurations, an offset of 50 mm preceding its third joint is present in the robot's links. This offset presented a choice: either to skew the subsequent frame about the z-axis or to incorporate an intermediary dummy frame prior to the offset, thereby eliminating potential computational anomalies. Our study explored both approaches to ascertain the accuracy of the resultant end effector pose. As anticipated, the EE poses derived from both conventions were congruent.

Figure 7 shows the frames, accompanied by the respective link lengths. Herein, the cylindrical representations correspond to the joints. Given the 5R joint configuration, all variables are exclusively joint angles, obviating the need for joint lengths, which simplifies the computational process. The depicted configuration represents the robot's neutral position, with all angles set to zero. This configuration is pivotal in understanding the frame assignments for each joint. It's noteworthy that frames 3 and 4 are superimposed; however, for clarity in representation, they are depicted distinctly.

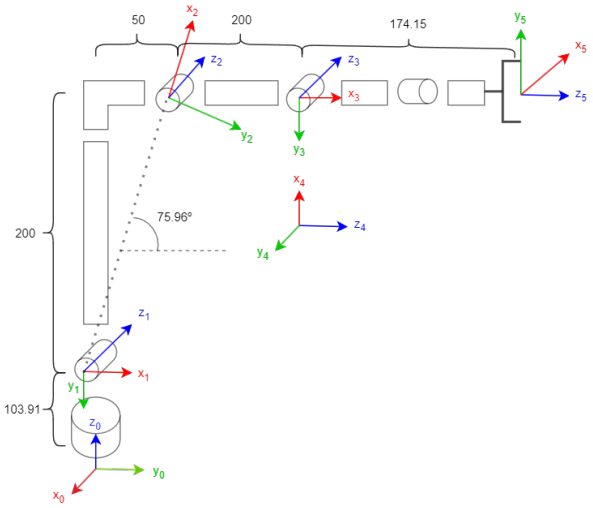

Fig. 7: kinematic visualization for FK

Subsequently, the DH parameters were computed and encapsulated within an array, poised for subsequent calculations. Each row of this array characterizes the pose of frame i with respect to frame $i-1$. the DH matrix is given below with the first row representing the parameters. Using the transformation matrix from equation 6, row on each row of equation 5, we can get a final transformation matrix between the base and end effector as T_5^0. The 4th column of this matrix gives a homogeneous 4 x 1 vector for the pose of End Effector.

$$DH = \begin{bmatrix} (\theta) & (d) & (a) & (\alpha) \\ t_1 + \pi/2 & 103.91 & 0 & -\pi/2 \\ t_2 - 1.3258 & 0 & 50\sqrt{17} & 0 \\ t_3 + 1.3258 & 0 & 200 & 0 \\ t_4 - \pi/2 & 0 & 0 & -\pi/2 \\ t_5 - \pi/2 & 174.15 & 0 & 0 \end{bmatrix} \tag{5}$$

$$T_{i-1}^{i} = \begin{bmatrix} c(\theta_i) & -s(\theta_i)c(\alpha_i) & s(\theta_i)s(\alpha_i) & a_i c(\theta_i) \\ s(\theta_i) & c(\theta_i)c(\alpha_i) & -c(\theta_i)c(\alpha_i) & a_i s(\theta_i) \\ 0 & s(\alpha_i) & c(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

The orientation for ZYZ or ZYX angles can be found using the rotation matrix embedded in the transformation matrix. We have found both angles but for future applications we used ZYX orientation for which the formula is straight forward and given below.

$$T_0^n = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$\begin{aligned} \text{Yaw (Z rotation, } \psi): &\quad \psi = \text{atan2}(r_{21}, r_{11}) \\ \text{Pitch (Y rotation, } \theta): &\quad \theta = \text{atan2}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \text{Roll (X rotation, } \phi): &\quad \phi = \text{atan2}(r_{32}, r_{33}) \end{aligned}$$

4 separate functions were used to compute everything. When asked, the joint angles and number of links were passed on to the $FK\_dh()$ function. It calculates the FK till the required link. Every successive transformation matrix is calculated using $get\_transform\_from\_dh()$ function multiplying matrices. After the final transformation is calculated, the Euler angles and pose are extracted using 2 separate functions and flattened to a 6 x 1 array which is passed back as an output.

Results and verification

After getting the theoretical pose and orientation, we displayed the values on the GUI. To verify the calculations, manual positioning of the robot arm was done where the End Effector was placed in known poses. Since the board's measurements are very precise (with absolute spacing of 25mm) the location calculated and actual EE pose has some difference. These ground truth values and calculated values were slightly off with a maximum combined offset reaching 10 mm in either axes. The possible reasons for the errors are listed below with 2 of them corrected during the deployment of inverse kinematics:

- *Joint Backlash*: Observed play or movement in a joint when its direction is reversed (Resolved).
- *Manufacturing Tolerances*: Imperfections in manufacturing affecting joint angles of servos.
- *Joint Offset Errors*: Observed errors in determining the zero position of a joint (Resolved).
- *External Loads*: Forces applied to the robot or The weight-induced sagging or bending in robot links.

B. Inverse kinematic solution

Inverse Kinematics (IK) is a fundamental concept in robotics that focuses on determining the joint parameters necessary to position the end effector at a specified location with a desired orientation. Unlike Forward Kinematics, IK computes the joint angles required to achieve a given end effector pose. The representation of orientation in IK can vary based on the specific requirements and complexities of the robotic system. For our station, implements the Yaw-Pitch-Roll representation

Our Approach

- *Determine Desired Pose*: Specifying the desired position for EE based on the competition tasks.
- *Choosing Orientation*: Decide on the orientation for the EE depending on the complexity of the tasks
- *Compute Joint Angles*: Utilize the IK function to compute the joint angles using analytical solutions.
- *Validation*: Reverse validation using FK. This compares both the theoretical calculations for FK & IK.

Our station employed the analytical approach where we decouple the 5 rotational joints to a 1R-2R-2R problem Let O_5 be the end effector frame. Provided the rotation matrix R_5^0 and the final link distance, we can move back to O_4 frame using the equation given below. From Fig. **??**, it is given that O_3 and O_4 coincide which means the frame O_3's location is not dependent on θ_4. Fig. 9 shows the decoupled first three joint angles. (x,y,z) are obtained from the frame O_4.

$$O_4 = O_5 - d_5 R_0^5 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{8}$$

It is obvious that θ_1 depends on the final EE pose which can always be calculated separately. We are then reduced to a 2R decoupled problem. A simplified geometric diagram is shown in Figure 4. As mentioned before the 'L' shaped link is replaced with its hypotenuse and an extra offset angle is always added for calculations which is the angle for the 'L' shape denoted by α. Two equations are formed from this diagram where θ_2 is purely determined from geometrical relations and θ_3 is dependent on geometry and θ_2

θ_3 can easily be found given the cosine rule. All the lengths of the triangle are available along with the angles

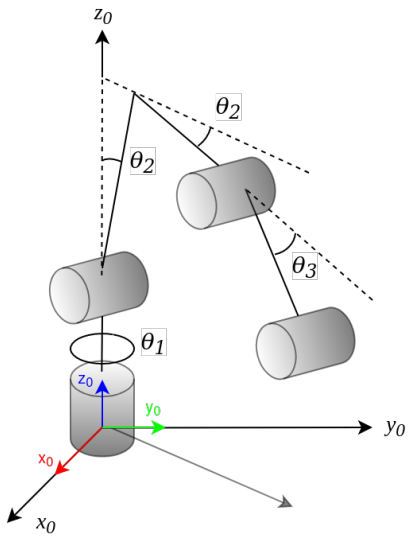

Fig. 8: Decoupling of RX200 arm

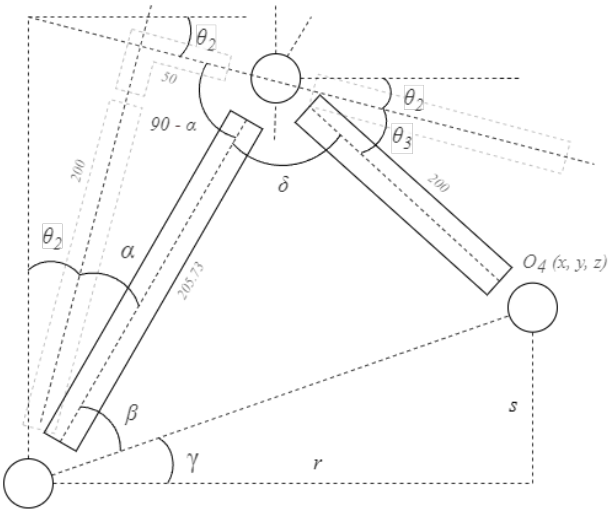

Fig. 9: Decoupled 2R kinematic chain

of the triangle. the equations for calculating θ_3 are given below.

$$\cos(\delta) = \frac{r^2 + s^2 - 200^2 - 200^2 - 50^2}{2 \times 200 \times 50 \times \sqrt{17}} \tag{9}$$

$$\delta = 180^o - \tan^{-1}(4) - \theta_3 \tag{10}$$

From the figure, it can be deduced that the equations for computing θ_2. β_2 can be found out using the sine rule in the inner triangle. which is dependent on θ_3. Due to this, we can get 2 possible pairs of (θ_2,θ_3). The equations for calculating θ_2 are given below.

$$\theta_2 = \frac{\pi}{2} - \alpha - \gamma - \beta \tag{11}$$

$$\beta = \sin^{-1}\left(\frac{200 \cdot \sin(\delta)}{\sqrt{r^2 + s^2}}\right) \tag{12}$$

$$\delta = 180^o - \tan^{-1}(4) - \theta_3 \tag{13}$$

From the equations above, 2 values of θ_2 can be obtained from geometric value of θ_3 resulting in 2 pairs of (θ_2,θ_3). Now based on EE pose, θ_1 can also vary between 2 values: 1 obtuse and 1 acute. Because of this, the value of θ_2 will change and hence θ_3. In conclusion, 4 sets of (θ_1,θ_2,θ_3) can be obtained. We have set a filter where the function will select the set where a maximum number of (θ_1,θ_2,θ_3) are acute angles.

After determining (θ_1,θ_2,θ_3), θ_4 and θ_5 can be determined using the rotation matrix R_4^5 obtain from equation below. Using a similar method as Z-Y-X angles, we now have the final 2 joint angles.

$$R_3^5 = R_0^{3^-}1 * R_0^5 \tag{14}$$

$$\theta_4 = \tan^- 1(\frac{R_3^5\ [1,2]}{R_3^5\ [0,2]}) + \frac{6\pi}{180} \tag{15}$$

$$\theta_5 = \tan^- 1(\frac{R_3^5\ [2,1]}{R_3^5\ [2,0]}) \tag{16}$$

For most of our tasks, the default orientation for the end effector is set to be perpendicular to the board and parallel to the board based on the requirements of the competition. If not reachable by these poses arbitrary position can be selected to pick up the blocks. (iterating from 0-180^o to find the minimum possible deviation from the neutral position to pick up the blocks). The complete algorithm for computing the joint angles for a given pose is given below.

Algorithm 1 Determine Joint angles using pose

Pass Pose, correction angles, and normal form to IK_geometric()
$r, s, O_4 \leftarrow normal\_form\_calculation()$
$determine\_formation(r, s, O_4)$
if θ_3 is successful **then**
  Calculate remaining angles
  Return Joint angles, Normal_form = True
else
  Calculate r, s, O_4 for horizontal orientation
  if θ_3 is successful **then**
    Calculate remaining angles
    Return Joint angles, Normal_form = False
  else
    Calculate the best pose with minimum deviation
    Return Joint angles

Validation and correction

To validate the calculations, we used random values to calculate the EE pose using the FK function and fed the resulting pose back to the IK function to regenerate the initial joint angles. We were able to generate angles with a precision of 0.3^o which proved the calculations correct.

During location validation, the robot consistently approximated the desired pose within its workspace, albeit with a deviation of 10-20 mm from the world coordinates and the actual position. This discrepancy was attributed to the system inaccurately reading the offset angles from the servos potentially due to loading or mechanical errors in the servos. To mitigate this issue, at the commencement of each task, joint values are recorded during initialization. These initialized angles are then utilized as correction factors; when the (IK) function is invoked, these angles are subtracted from the calculated angles to align the robot's position with the correct world coordinates. This adjustment significantly reduced the deviation to approximately 5 mm.

C. Path Planning

An algorithm is put in place for the robot's motion planning for block manipulation. Each time a point is supplied for the robot to manipulate, the robot creates 3 waypoints as described in algorithm 2. The robot will navigate to a position above the block, go down to grab or release the block, and raise itself up again. This function ensures the robot picks and places objects without colliding with the ground or reaching an incorrect position due to inertial forces. Through experimentation, it was determined a reasonable offset for the $aboveheight$ and $intermediateheight$ values is 100mm from the top of the block. This helps avoid knocking other blocks out of position blocks and gives the robot enough space to maneuver. This algorithm is consistently employed across all competition tasks to compute waypoints after determining the end effector (EE) pose based on the required reasoning. After appending, The variable is passed to move the arm. The algorithm for motion planning is given below.

Algorithm 2 Motion Planning for a set pose

Data: $P_w(x, y, z)$, grab, angle(ψ), is_large
$\psi = \frac{(90-\psi)}{180} \cdot \pi$
Define above_height and intermediate_height
Determine z offset based on size(is_large) and action(grab)
$z =$ location[2] + offset
for phase in [prepare, grab, intermediate, above] **do**
  $(\psi, \text{form}) \leftarrow$ IK_geometric($P_w, P_w^{init}, \psi, \text{form}$)
  Update taught_waypoints with new ψ, form, and action (grab or release)

IV. Results

Event 1: Pick n' Sort

This task involved automatically segregating large and small blocks. Leveraging previously defined functions, we developed an algorithm for this task (refer to Appendix). The algorithm successfully segregated the blocks into two piles in a level 3 difficulty scenario with 9 random blocks, completing the task within the allotted time. Although the persistent issue of not being able to pick and place blocks accurately prevented the two block piles from being clean, the segmentation was nonetheless successful. The overview of the method used to accomplish this task is included in the appendix as algorithm 3.

Event 2: Pick n' Stack

For the next task, the arm was tasked with stacking blocks on Apriltags. In a level 3 difficulty setting, it successfully stacked 3 small and 3 large blocks on 3 small blocks on the Apriltags in approximately 1.5 minutes. Despite occasional toppling of the small stack, the algorithm was successful 90% of the time. The overview of the method used to accomplish this task is included in the appendix as algorithm 4.

Event 3: Line 'em up

Task 3 focuses on arranging the different colored blocks (both small and large) into the 'rainbow' order. The algorithm faced some limitations due to the possibility of locks in the process. Locks occur when all predetermined valid placement locations are blocked by other color blocks. The algorithm is not able to resolve it and simply terminates prematurely. This issue is exacerbated by the inaccuracy of picking and placing blocks which sometimes causes a block to occupy multiple spaces. This also prevents reliability in being able to line the blocks in a perfect line. Ultimately although the algorithm is able to perform at level 3, it is too unreliable and inconsistent to achieve perfect points. The overview of the method used to accomplish this task is included in the appendix as algorithm 5.

V. Discussion

In robotic arm tasks, there are many points where errors can accumulate and there are countless sources of error that cannot be accurately modeled. In the pipeline from block detection to the end effector movement, major sources of error that reduced the accuracy of the manipulation tasks include computer vision noise, coordinate transformation errors and mechanical errors. This ultimately led to large inaccuracies in block grabbing and placing. The most common issue was inaccurate z-position when gripping and releasing blocks. This issue was most pronounced in task 2 where z-position errors when placing the block can cause blocks to fall off the stack or even topple the stack over. It also presented issues when placing blocks down in task 1 and task 3. Movement inaccuracies prevent the gripper from accurately grabbing the block and the error is further propagated when placing the block down. This caused the arm's inability to build a perfect line.

Computer vision is also a challenging task for robotic arms. In this project, pure color thresholding-based block detection was used. The issue with this is that the algorithms are less robust when responding to edge cases. For example, when dealing with the situation of a small block stacked on a large block, the algorithm will detect 2 blocks overlapping each other and can cause undefined behavior. Large amounts of extra work were dedicated to compensating for computer vision edge cases like these which made the programs inefficient and limited.

There is also an unexpected error associated with inverse kinematics that caused problems with performing the tasks. In certain cases, the desired end effector position can be unsolvable with the IK equations. This causes the robot to stall and the behavior is undefined. The solution for this issue is add incremental waypoints which will significantly slow the arm down when trying to carry out tasks.

VI. Conclusion

In summary, by understanding the robot's dynamics and camera properties, we successfully automated the RX200 arm for defined tasks. The difference between manual and Apriltag calibration highlighted the need for precise pixel-to-real-world correspondence. After calibration, we processed the input to categorize blocks, differentiating by size, color, and removing irrelevant distractor blocks. For robot movement towards detected blocks, both forward and inverse kinematics were employed. Though theoretical forward kinematics showed minor discrepancies from the ground truth, corrective measures minimized errors to under 5 mm. With the joint configuration determined, traversal path planning was essential to avoid collisions and optimize task efficiency. Overall, our efforts led to Level 3 Automation of the robot arm for predefined tasks.

References

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

[2] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

VII. Appendices

A. Algorithms for reference

Algorithm 3 Pick n' Sort

```
Check camera calibration.
if camera is uncalibrated then
    Exit with an error.
else
    Initialize the robotic arm.
    while not is_finished do
        Clear previous waypoints.
        Sort blocks using block_size_sort().
        if small_blocks, large_blocks are empty then
            Set is_finished flag to true.
        Identify placement locations.
        for block in small_blocks, large_blocks do
            Add pick and place waypoints.
        Execute the waypoints.
    end while
    Put the robotic arm to sleep.
    Display total time taken for the task.
```

Algorithm 4 Pick n' Stack

```
Initialize target locations
Initialize robot arm
for each block size from small to large do
    while blocks of current size remain do
        Determine pick and place locations
        Execute pick and place operations
        Capture picture of the current state
    end while
```

Algorithm 5 Line 'em up

```
while valid moves remain do
    Identify the location of a movable block by color.
    Check destination for obstacles.
    if non-color blocks or distractor blocks are present
then
        Remove these obstacles.
    else if destination is clear then
        Move the block to the destination.
    else if another color block occupies the destination
then
        Skip the block.
end while
```

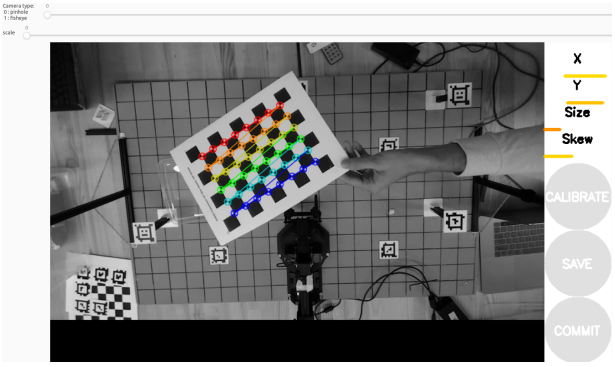

Fig. 10: Application for calculating the intrinsic matrix

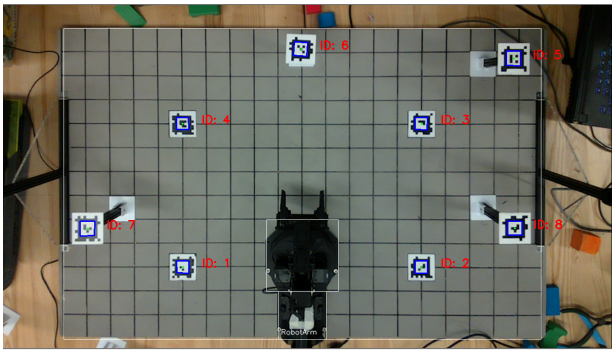

Fig. 11: Region of Interest

Fig. 12: HSV and YCrCb color space tuning