# ROB 550 Armlab Report

Kuan-Ting Chen, Emaad Gerami, Ziyu Guo
{kthchen, egerami, ziyuguo}@umich.edu

***Abstract*—For robots to be able to perform autonomously in the real world, they must be able to observe their environment and be able to determine the most optimal action based on what it is able to detect. To do so, the robot must be able to properly sense, reason, and act based on what it sees. We are proposing a computer vision algorithm and a path planning algorithm to allow an RX200 arm to act autonomously to move wooden blocks. To do so, a RealSense L515 RGB-D camera is used to provide color and depth images of the robot's environment. From these images, we can extract useful information such as 3D positions, orientation, color, and size of the blocks in the workspace. This information is then passed to the path planning algorithm, which uses forward and inverse kinematics to calculate the most optimal trajectory for the RX200 robotic arm to follow based on the task it has been told to complete. Our approach was partially able to solve tasks in a competitive environment. With more time, the robot would be able to complete all tasks presented before it.**

Fig. 1: The RX200 robotic arm

Fig. 2: The RealSense L515 RGB-D camera

## I. Introduction

Robot manipulation can mainly refer to the ways in which robots can interact with the objects around them, and is one of the budding aspects of robotics in society. While robots can be operated by a person telling the robot what actions to perform, it is also possible for a robot to complete those same actions completely autonomously without any intervention from an operator. In order for a robot to act autonomously, it must be able to complete three different sub-tasks in order to operate. The first sub-task is sensing, which refers to detecting the objects the robot will interact with using touch sensors, computer vision, or any other types of sensors. The next sub-task is reasoning, which involves the robot manipulating its own internal model of the world around it to determine which actions it could perform. The robot then completes its final sub-task, acting, in which the robot completes its main task.

The main problem that our group set out to solve was to allow the RX200 (Fig. 1), a robotic arm with five degrees of freedom, to autonomously manipulate wooden blocks placed in the robot's workspace. In order to see where the blocks were located, a RealSense L515 RGB-D camera (Fig. 2), positioned above the robot's workspace at a slight rotational offset, was used in order to detect the location, rotation, size, and color of the blocks located in the robot's workspace and relay the necessary information to the robot in order to allow the robot to grab the blocks and place them without much difficulty or trouble.

Firstly, the methodology (Section II) will be discussed, as it will explain what was needed in order for the robot to detect the blocks in the robot's workspace in order to grab and place them. The robot's performance (Section III) will then be described, followed by the discussion (Section IV), where it will be elaborated as to which aspects of the robot's manipulations were successful and which aspects underperformed. Lastly, the conclusion (Section V) will provide a brief summary of what was discussed in the paper and explain what could potentially be improved if the team had more time to work on the project.

## II. Methodology

### A. Camera calibration

The intrinsic matrix consists of the optical center (cx, cy) and focal length (fx, fy) of a camera. The matrix is unique to a specific camera, so once calculated, it can be reused on other images taken by the same camera. To obtain intrinsic parameters, we perform intrinsic calibration using the ROS intrinsic calibration tool. These parameters are verified by comparing them with results using the factory calibration.

$$\begin{bmatrix} f_x & 0 & C_x \\ 0 & f_y & C_y \\ 0 & 0 & 1 \end{bmatrix}$$

The intrinsic matrix describes a projective transformation from the 3D camera frame to the 2D image frame. The matrix is crucial for our project as we use images to reconstruct the workspace. Because we assumed the errors in depth values from the camera were negligible, so we did not perform depth calibration.

Intrinsic calibration estimates the camera parameters by using images that contain a calibration pattern. We found the intrinsic matrix for the RealSense camera by using the ROS camera calibration tool and an 8x6 checkerboard. We moved the checkerboard around in front of the camera in order to cover as many possible locations and orientations of the checkerboard as possible. In order to obtain an accurate calibration, we repeated the calibration procedure five times to obtain an average result. To get the factory calibration parameters, we used the *rostopic echo /camera/color/camera_info* command in ROS.

### B. Workspace Reconstruction

We rely on extrinsic parameters to reconstruct the workspace. The extrinsic matrix consists of a rotation R and a translation t. It represents a rigid transformation from the 3D world frame to the 3D camera frame. A simple method of estimating the camera's extrinsic parameters is to use the physical measurements of the workspace, but it can only provide rough estimation. 4 Apriltags placed on the board were used to provide data points for estimating the extrinsic matrix. We used their 2D and 3D positions as inputs for the OpenCV solvePnP function and solved for an extrinsic matrix. With intrinsic and extrinsic parameters, we can reproject 2D pixels to the world frame in order to obtain 3D positions.

To estimate the extrinsic matrix of the camera, we need a 2D to 3D correspondence in the image frame and the world frame. The 4 Apriltags placed on the board were used. The world coordinates of the Apriltags were obtained by manual measurements, while their pixel coordinates were obtained by mouse clicks. From here, the OpenCV perspective-n-point (PnP) algorithm is used to estimate the rotation and translation between the world frame and the camera frame. Note that the rotation returned from the PnP function is a rotation vector in axis-angle form, so Rodrigues' formula is used to convert a rotation vector to a rotation matrix. We then combine the rotation and translation into a 4x4 matrix $\in$ SE(3) representing the extrinsic matrix.

To obtain the 3D positions in the world frame, the pixel coordinates are transformed into the camera frame ($X_C$, $Y_C$, $Z_C$) by multiplying an inverse intrinsic matrix. With depth information, we can undo the scale by multiplying $Z_C$.

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c(u,v)\mathbf{K}^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

The points in the camera frame are expanded to homogeneous coordinates and multiplied by an inverse extrinsic matrix to obtain the 3D positions in the world frame.

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \mathbf{H}^{-1} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{2}$$

### C. Block detection

The block detection algorithm is designed to extract geometric information about the top block of each stack from the depth image. Since the workspace is flat, locating blocks in the depth image is quite simple as long as a proper threshold of depth value is set. We then created contours by binding the top surfaces of each block. These contours are used to find the centroids, size, and angles of the blocks. A solution to color detection is to use the RGB values to compute the Euclidean distance of each known color and find the color with the nearest RGB value to the block. We assumed that there were only six colors of blocks: 'red', 'orange', 'yellow', 'green', 'blue', and 'violet'. However, the RGB color space is not suitable for color detection because RGB values are subject to brightness, color intensity, and lighting variations within the workspace, so we instead used the HSV color space, which separates the color information to the hue channel (Fig. 3).

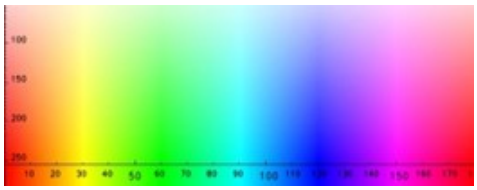

Fig. 3: Hue Spectrum in the HSV color space

Block detection involves finding the 3D positions, angles, sizes, and colors of blocks located in the workspace. First, we used the depth image to detect the block's geometry. We then applied a bilateral filter with a 5x5 kernel to smooth the depth image as the depth data from the RealSense camera was fairly noisy. We decided to use the bilateral filter because it is designed to reduce noise in an image while simultaneously preserving edges. By defining the workspace and masking the arm,

we can then localize the blocks by thresholding the depth image. In addition, erosion and dilation were applied to reduce the chance of detecting a false positive. Note that the order of erosion and dilation cannot be reversed.

The depth threshold creates a few contours in the image. The centroids of these contours are calculated by the moments function in OpenCV. We then bound the contours with the OpenCV minAreaRect function, which returns the angle with respect to the x-axis of the image. Since we have already determined the intrinsic and extrinsic matrices, we can reproject the centroids to the world frame in order to obtain the 3D positions. Lastly, to determine the size of the block, we compute the area of each contour. If the area is larger than 1000, we label the block as large.

We utilized the RGB image to detect the color of the block, but we opted for HSV color space since it separates the color information to the hue channel. In addition, the hue channel follows the order of the colors in a rainbow. To find the dominant color of a contour, we built a histogram of 180 bins and calculated the most frequent hue value. We then determined the block color by finding the nearest hue values in our dictionary.

### *D. Forward Kinematics*

Forward kinematics (FK) is a frequently used technique in robotics that involves computing the orientation and position of the end-effector or the joints on the robot arm. To summarize, given the positions and orientations of the joints on the robot arm, we are able to use FK to calculate the pose and location of the end-effector. The FK process uses a set of transformation matrices to move from the base of the robot to the end-effector. By multiplying the transformation matrices together, we can get the overall transformation matrix. Due to the ability to estimate the status of the end-effector, FK can be especially useful in motion planning.

Generally, the two main methods for FK are the Denavit-Hartenberg (DH) convention method and Product of Exponential (PoE) method. In our work, we decided to use the DH convention method. The DH convention defines a standard set of coordinate frames and joint parameters that are used to model the kinematic chain of the robotic system.

We first assign all the frames on the robot arm by strictly following the rules of assigning frames (see Fig. 4).

To express the transformation from one frame to another, we drew a DH table that contained all the DH parameters, including the link length $a_i$, link twist $\alpha_i$, joint offset $d_i$, and joint angle $\theta_i$ (see TABLE 1). The DH table provides a way to describe the kinematics of a robotic system in a simple and consistent manner, which makes it easier to compute FK.

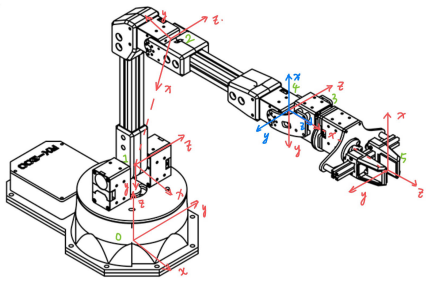

Fig. 4: Frame Settings for the RX200 Arm

| DH Params or Link | $\theta_i$ | $d_i$ | $a_i$ | $\alpha_i$ |
|---|---|---|---|---|
| 1 | $\theta_1$ | 103.91 | 0 | $-\pi/2$ |
| 2 | 104.03°+$\theta_2$ | 0 | -205.73 | 0 |
| 3 | -104.03°-$\theta_3$ | 0 | 200 | 0 |
| 4 | $-\pi/2$+$\theta_4$ | 0 | 0 | $-\pi/2$ |
| 5 | $\theta_5$ | 174.15 | 0 | 0 |

TABLE I: DH Table

Each row in the DH table represents a homogeneous transformation matrix which is constructed as (3). $A_i$ is the homogeneous transformation matrix of row i.

$$
\begin{aligned}
A_i &= \mathrm{Rot}_{z,\theta_i}\,\mathrm{Trans}_{z,d_i}\,\mathrm{Trans}_{x,a_i}\,\mathrm{Rot}_{x,\alpha_i} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned} \tag{3}
$$

The complete FK equation is assembled from the individual homogeneous transformations generated from each row in the DH table. In (4), H can be decomposed into the rotation matrix $R_n^0$ and the translation matrix $o_n^0$.

$$
H = A_1(q_1)\ldots A_n(q_n) = \begin{bmatrix} R_n^0 & o_n^0 \\ 0 & 1 \end{bmatrix} \tag{4}
$$

After calculating the complete homogeneous transformation matrix, we are able to compute the location and orientation of the end-effector using joint angles. In our work, the end-effector position is defined by x,y, and z in (6), and $\phi$, $\theta$, $\psi$ in (5), where the orientation uses the common ZYZ Euler angle convention.

$$
\begin{aligned}
R_{ZYZ} =& R_{z,\phi} R_{y,\theta} R_{z,\psi} \\
=& \begin{bmatrix} c_\phi c_\theta c_\psi - s_\phi s_\psi & -c_\phi c_\theta s_\psi - s_\phi c_\psi & c_\phi s_\theta \\ s_\phi c_\theta c_\psi + c_\phi s_\psi & -s_\phi c_\theta s_\psi + c_\phi c_\psi & s_\phi s_\theta \\ -s_\theta c_\psi & s_\theta s_\psi & c_\theta \end{bmatrix} \qquad (5)
\end{aligned}
$$

$$
o_n^0 = \begin{bmatrix} x & y & z \end{bmatrix}^\top \qquad (6)
$$

By comparing $R_n^0$ in (4) with $R_{ZYZ}$ in (5), we can compute $\phi$, $\theta$ and $\psi$ using the equations in (7)

$$
\begin{aligned}
\theta &= \arccos\left(R_n^0[3,3]\right) \\
\phi &= \arcsin\left(\frac{R_n^0[2,3]}{\sin\theta}\right) \\
\psi &= \arcsin\left(\frac{R_n^0[3,2]}{\sin\theta}\right)
\end{aligned} \qquad (7)
$$

Therefore, from (6) and (7), we get the location and orientation of the end-effector.

### *E. Inverse Kinematics*

Inverse kinematics (IK) involves calculating the joint angles required for the end-effector of the robot arm in order to achieve an ideal location and pose, being the inverse of forward kinematics. Given the desired position and orientation of the end-effector, we are able to get the angles of each joint. The two IK solutions are the closed-form solution and the numerical solution. The closed-form method uses mathematical equations to derive closed-form solutions for IK problems. The numerical method uses numerical optimization techniques to refine an initial guess for the joint variables, which usually requires a large computing power. For our robot arm, we choose to use the closed-form solution, by using geometrical methods to get the joint angles.

Several methods, including analytical solutions and numerical solutions, are used to solve IK problems. Analytical solutions exist in unique cases, such as 6-DoF robots. To apply the analytical method, we can use geometric relationships and kinematic decoupling methods to analyze and simplify the problem, which is computationally efficient. While the numerical method relies on iterative optimization to seek out an approximate solution, it also requires a large amount of computing power. In our work, due to the robot arm being simple to analyze, we used the analytical method to solve the IK problem.

Generally, positioning the end-effector either horizontally or orthogonally should always be the safest configuration when grasping and dropping a block on the plane. Therefore, we can simplify the problem by only considering two situations, where the end-effector is horizontal (Fig. 5) or orthogonal (Fig. 6).

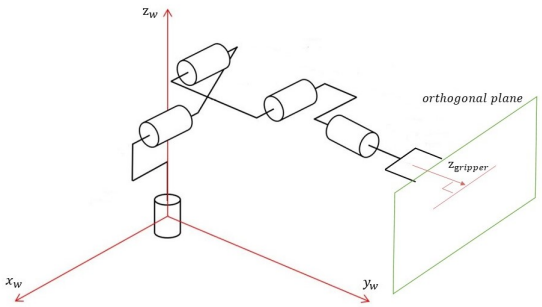


Fig. 5: Illustration for Horizontal Reach

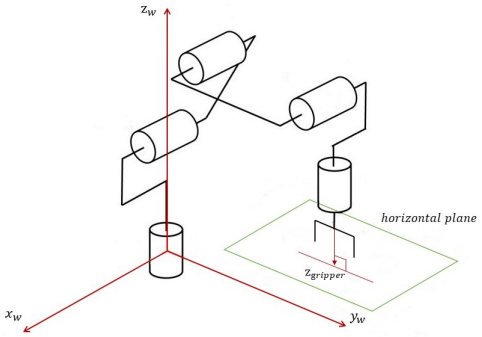


Fig. 6: Illustration for Orthogonal Reach

Since the angle of joint 1 ($\theta_1$) is directly determined by the block position and joints 2, 3, 4, and 5, along with the end-effector, are all in the same plane (see Fig. 7), we are able to simplify the 3D geometrical problem into a 2D geometrical problem.

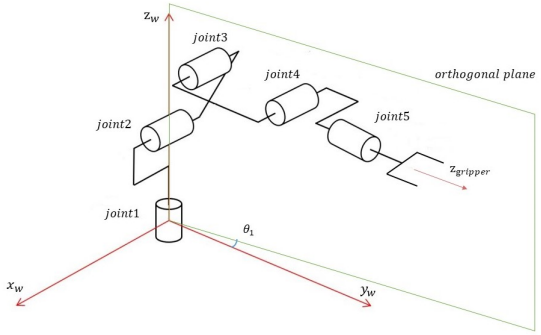


Fig. 7: Simplify a 3D Geometrical Problem into a 2D Geometrical Problem (Horizontal Reach Case)

$\theta_1$ is defined by the location of the end-effector. In (8), $x_{ee}$, $y_{ee}$ and $z_{ee}$ represent the location of the end-effector in the world frame.

$$
\theta_1 = -\arctan 2\left(x_{ee}, y_{ee}\right) \qquad (8)
$$

Taking into consideration that joints 2, 3, 4, and 5

are at the same orthogonal plane, we solve the problem at this frame. For the horizontal reach situation, the 2D figure is shown in Fig. 8. For the orthogonal reach situation, the 2D figure is shown in Fig. 9. A, C, D, and E indicate joints 2, 3, 4 and the end-effector. $\theta_2$, $\theta_3$, $\theta_4$ are the joint angles of joints 2, 3, and 4. X and Y represent the horizontal distance and orthogonal distance between A and D. And $l_1$ - $l_5$ are the links between each joint. $l$ is the line AD, and $l_6$ is the line AC.

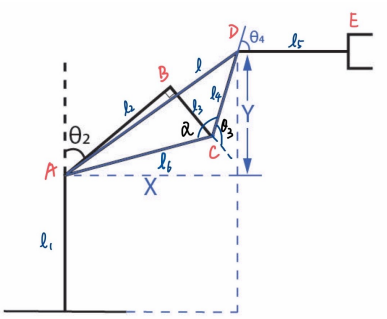


Fig. 8: 2D View for Horizontal Reach

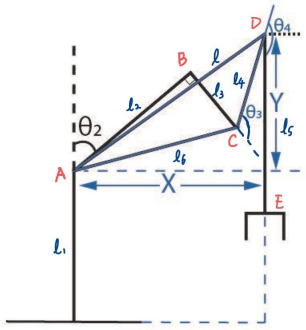


Fig. 9: 2D View for Orthogonal Reach

We first calculate the horizontal distance and orthogonal distance between A and D for both cases. X and Y in the horizontal reach case are computed in (9), while X and Y in the orthogonal reach case are computed in (10).

$$X = \sqrt{(x_{ee} + l_5 \sin\theta_1)^2 + (y_{ee} + l_5 \cos\theta_1)^2} \tag{9}$$
$$Y = z_{ee} - l_1$$

$$X = \sqrt{x_{ee}^2 + y_{ee}^2} \tag{10}$$
$$Y = z_{ee} + l_5 - l_1$$

Next, we calculated $\alpha$ and the joint angles $\theta_2$, $\theta_3$, $\theta_4$, $\theta_5$. Numpy.arctan2 is a function included in the Numpy package in python. It returns the element-wise arc tangent of x1 and x2 by choosing the correct quadrant.

$$\begin{aligned}
\alpha &= \arccos \frac{\left(X^2 + Y^2 - l_4^2 - l_6^2\right)}{2 l_4 l_6} \\
\theta_3^1 &= \alpha + \arctan 2\left(l_2, l_3\right) \\
\theta_3^2 &= -\alpha + \arctan 2\left(l_2, l_3\right) \\
\theta_2^{1'} &= \arctan 2(Y, X) - \arctan 2\left(l_4 \sin\alpha, l_6 + l_4 \cos\alpha\right) \\
\theta_2^{2'} &= \arctan 2(Y, X) + \arctan 2\left(l_4 \sin\alpha, l_6 + l_4 \cos\alpha\right) \\
\theta_2^1 &= \frac{\pi}{2} - \theta_2^{1'} - \arctan 2\left(l_3, l_2\right) \\
\theta_2^2 &= \frac{\pi}{2} - \theta_2^{2'} - \arctan 2\left(l_3, l_2\right)
\end{aligned} \tag{11}$$

For the horizontal reach case:

$$\begin{aligned}
\theta_4^1 &= -\left(\theta_2^{1'} + \alpha\right) \\
\theta_4^2 &= -\left(\theta_2^{2'} - \alpha\right)
\end{aligned} \tag{12}$$

For the orthogonal reach case:

$$\begin{aligned}
\theta_4^1 &= -\frac{\pi}{2} - \left(\theta_2^{1'} + \alpha\right) \\
\theta_4^2 &= -\frac{\pi}{2} - \left(\theta_2^{2'} - \alpha\right)
\end{aligned} \tag{13}$$

Finally, we calculated $\theta_5$. In the horizontal reach case, $\theta_5$ is set to be equal to 0 at all times. While in the orthogonal reach case, $\theta_5$ is related to the block angle (see Fig. 10)

If $\theta_1 \geq 0$ and $\theta_1 \leq \pi/2$:

$$\theta_5 = \theta_1 - \psi \tag{14}$$

Else if $\theta_1 > \pi/2$:

$$\theta_5 = \theta_1 - \psi - \frac{\pi}{2} \tag{15}$$

Else if $\theta_1 \leq 0$ and $\theta_1 \geq -\pi/2$:

$$\theta_5 = \theta_1 - \psi + \frac{\pi}{2} \tag{16}$$

Else,

$$\theta_5 = \theta_1 - \psi + \pi \tag{17}$$

We had finished the calculation of the joint angles for both cases, and after testing on the robot arm, we selected one appropriate solution:

$$\text{angles} = \begin{bmatrix} \theta_1 & \theta_2^2 & \theta_3^2 & \theta_4^2 & \theta_5 \end{bmatrix} \tag{18}$$

To determine the method used to reach the object, we added some constraint conditions to let the robot arm automatically decide the most optimal approach method. If AC, CD, and AD can form a rectangle and $X = \sqrt{x_{ee}^2 + y_{ee}^2} \leq 350$ and $z_{ee} \leq 350$, the robot arm would decide to reach orthogonally, reaching horizontally if that is not the case.

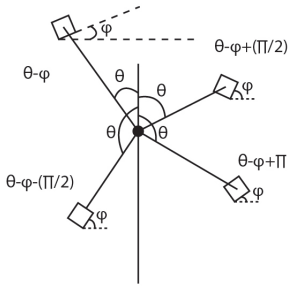


Fig. 10: Orthogonal reach at different positions

### F. Path Planning

The two main tasks that our robot needed to complete were to grab and place wooden blocks. In order to ensure that the robot could complete these two tasks, we came up with some paths for the robot to take that would not interfere with the blocks placed around the environment.

The first part of the robot's motion to grab a block was to move to a position located 30cm above where a block was detected in order to allow the end-effector to adjust its orientation to grab the block in the most efficient manner. If the block was within the robot's range, the end-effector would rotate to a orthogonal configuration to secure the block with increased precision. Otherwise, the end-effector would stay in the horizontal configuration. From there, the arm would descend to a position 10cm above the block, before moving down to a position where the end-effector would be aligned with the center of the block. The gripper would then close, securing the block. Once the block is secure, the arm would return to the waypoint located 30cm above the block's former position.

To place the blocks, the robot would take the block to a designated waypoint located 30cm above the block's final position. While moving to this waypoint, the robot would determine if the block could be placed at the desired location in the orthogonal configuration. If this was not the case, the arm would instead shift to the horizontal configuration and move to the waypoint. Once in the proper configuration, the robot would descend to 10cm above the location before becoming level with the board and opening its gripper before returning to the 30cm waypoint above the block's final location.

When stacking blocks, the end-effector's horizontal configuration is required for taller stacks due to the orthogonal configuration's inability to stack a large number of blocks. In addition, the arm moved 30cm above the stack after placing the block to ensure that it did not come into contact with the stack, preventing the risk of knocking over other blocks when moving to perform its other actions.

### G. State Machine

We wrote out the functions for each of the checkpoint tasks, including the competition events:

1) *Teach and Repeat:* We recorded the joint angles of the robot at the desired waypoints in a list using a button in the GUI. Once recorded, the robot would then iterate each of the waypoints in the list into forward kinematics in order to implement a complete cycle.
2) *Autonomy:* In order to grab stacked blocks, we implemented an addition to our detect function to determine if a block was stacked. At the beginning of each event and subsequent loops, the robot initially checked to see if there were any blocks in the general height range of a stack of four large blocks, as that was the maximum height of any pre-set stacks we would be dealing with and was determined by measuring the average height of a large block. If any blocks were detected, the robot would grab and place them first. If there were no blocks detected in the height range, the robot would then check the range of a stack of three large blocks, grabbing any detected blocks and placing them. This process would then repeat for any stacks of two large blocks or regular blocks. This addition would allow us to ensure that the robot could effectively handle any stacks that it would need to face, and would additionally be able to detect and place stacks of small blocks.
   In addition to prioritizing stacked blocks, the robot would also grab blocks that were closer to the robot's base, as if the robot would run the risk of running into other blocks placed on the board when attempting to grab farther blocks first. The blocks were also placed in the negative-half plane before being stacked, as the detect functions would not check the negative-half plane for blocks, so they could be placed there without having to worry about any issues with placement.

## III. Results

### A. Camera calibration

We repeated the calibration procedure using the ROS calibration tool five times. The average intrinsic parameters are:

$$K_1 = \begin{bmatrix} 920.073 & 0 & 658.054 \\ 0 & 923.201 & 376.957 \\ 0 & 0 & 1 \end{bmatrix} \quad (19)$$

The factory calibration result obtained from *camera_info* in ROS is:

$$K_2 = \begin{bmatrix} 902.187 & 0 & 662.349 \\ 0 & 902.391 & 372.227 \\ 0 & 0 & 1 \end{bmatrix} \quad (20)$$

Compared with the factory calibration, we found that the optical center is accurate. However, the estimated focal length was off by 2.1%.

### B. Workspace reconstruction

We performed extrinsic calibration with both physical measurements and automatic calibration. The physical calibration result was:

$$H_1 = \begin{bmatrix} 1 & 0 & 0 & 33 \\ 0 & -0.9656 & -0.2588 & 165 \\ 0 & 0.2588 & -0.9656 & 986 \end{bmatrix} \quad (21)$$

The automatic calibration result was:

$$H_2 = \begin{bmatrix} 0.9951 & -0.0265 & -0.0126 & 44.59 \\ -0.0240 & -0.9855 & 0.1676 & 173.40 \\ 0.0169 & 0.1672 & -0.9857 & 1033 \end{bmatrix} \quad (22)$$

The two results above are similar with noticeable errors in the translation. To verify the correctness of the extrinsic calibration, we projected a grid of points representing the corners of the workspace onto the image. Additionally, we applied the perspective transform to shift the perspective of the image to appear as a birds-eye view.

Fig. 11: Projection of grid points using the intrinsic and extrinsic matrix

$$homography = \begin{bmatrix} 1.032 & -0.137 & -72.940 \\ 0.029 & 0.997 & -80.911 \\ 0 & 0 & 1 \end{bmatrix} \quad (23)$$

### C. Block detection

Our block detection algorithm performed well overall. However, there were still a few problems. Blocks angles were not accurate enough because contours could contain noise. A color misclassification occurred where small violet blocks were sometimes classified as small yellow blocks.

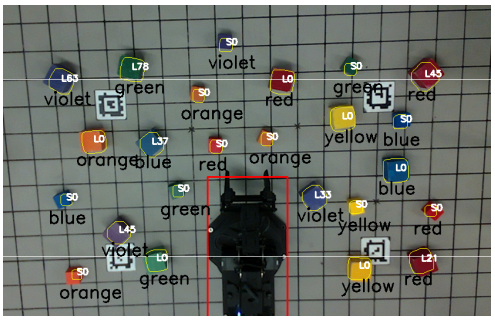


Fig. 12: Block detection result. L and S represent size; Numbers represent block angles with respect to the x-axis.

### D. Teach and Repeat

We taught our robot to cycle two blocks between the locations (-100, 225) and (100, 225) through an intermediate location at (250,75), and the robot was able to cycle 10 times before being manually stopped. While we had a plot for joint angles over time for one cycle, it was unfortunately lost when our lab computer was unexpectedly wiped before we had a chance to pull the image, and could not be recovered, despite our best efforts.

### E. Autonomy

Event 1: We were able to properly grab and place all the blocks based on their size within the 180-second time limit, allowing us to earn the full 300 out of 300 points.

Event 2: We were able to fully stack all the blocks, both large and small, in practice, but for the actual competition we were unable to properly run the event, earning us 0 points.

Event 3: We lined up all six big blocks correctly and lined up 5 of the small blocks properly while still remaining fairly organized and neat, earning us 344 out of 400 points.

Event 4: We were able to stack the large blocks in rainbow order in practice, but due to an oversight on our part, we were unable to stack any of the blocks in the actual competition and received 0 points.

Bonus Event: While we were unable to complete the event in time for competition, we could stack a total of 8 blocks when tested afterward.

## IV. Discussion

### A. Camera Calibration

Compared to the factory calibration, we found that the optical center was accurate, but the focal length was off by 2.1 percent. This may have been caused by spatial limitations, as the camera had a fixed position. To improve our results, we would consider removing

the camera from its fixed location and collecting more samples at a farther distance. The other possible source of error in our calibration could be caused by the checkerboard not being completely flat.

### B. Workspace Reconstruction

The estimated extrinsic parameters were precise as shown in our results. This enabled the robotic arm to perform a variety of tasks that required spatial information. Our arm could pick up blocks consistently at any location. However, there is still room for improvement where the RGB and depth images could be aligned by performing an affine transformation.

### C. Block detection

Our block detection algorithm performed well overall but sometimes struggled to differentiate between violet and yellow objects, often recognizing small violet blocks as small yellow blocks. We also observed that the contours were a little off in some cases, which may have been caused by an unalignment of the RGB and depth images. In addition, if the blocks were placed close together, our algorithm had a chance to potentially be unable to differentiate the blocks. To fix this, color images and depth images could both be used.

### D. Kinematics

The position accuracy of the Inverse Kinematics was tested using the following method. For each reach mode, we input the 6 target coordinates. After the robot arm reached a target, the actual joints were recorded from the ROS topic: /rx200/joint states, which were put into a forward kinematics function to obtain the actual end-effector coordinates. We used the L2 norm difference to calculate the error:

$$\text{error} = \sqrt{(x_{\text{target}} - x_{\text{real}})^2 + (y_{\text{target}} - y_{\text{real}})^2 + (z_{\text{target}} - z_{\text{real}})^2} \tag{24}$$

The results are shown in TABLE II:

| Reaching Mode | Avg. Error $(\text{mm})$ |
|---|---|
| Horizontal Reach | 6.3411 |
| Orthogonal Reach | 3.1560 |

TABLE II: Average Error for Each Reaching Mode

### E. Autonomy

An issue that we faced at competition was our lack of any way to automatically generate move time and acceleration time parameters based on the robot's next waypoint. Due to an oversight in our programming, this step was overlooked, so it was never implemented in time for competition, as we instead manually set move time and acceleration time parameters for each individual event to a preset value.

Event 1: Even though we were able to earn the full 300 points for this event, we could potentially make changes to where the blocks were placed in the negative-half plane, which could potentially mean that the robot would never need to shift to the horizontal configuration when placing blocks and instead remain in the orthogonal configuration.

Event 2: When programming for the event, we did not account for small blocks being stacked on top of large blocks, so when we attempted this event at competition, our code was not properly adjusted to account for it. With more time, we could account for this oversight by placing the small blocks on the left side of the negative-half plane before stacking the large blocks and then stack the small blocks on top of the large blocks.

Event 3: While we were unable to line up all the blocks due to issues caused by the block detection, the event could also be improved to place the blocks closer together and in a more organized configuration than what we were able to complete at competition.

Event 4: Even though the robot was able to stack all the large blocks, it could not stack any of the small blocks. If we had more time, we would make sure the robot could improve our accuracy for the small blocks, allowing us to earn full points for the event.

Bonus Event: While we could stack 8 blocks outside of the competition, the number of blocks that the robot could stack was incredibly inconsistent. Therefore, if we had more time to experiment, we would ensure that the robot could consistently stack 8-10 blocks.

## V. Conclusion

We implemented a computer vision algorithm and path planning algorithm to our robot to allow it to autonomously grab and place wooden blocks. Our proposed computer vision algorithm enabled the arm to sort and stack blocks of various sizes and colors, utilizing efficient block detection and accurate color detection, although there were some errors. The path planning algorithm allowed the arm to find an efficient path to each of the blocks, while having a few inaccuracies when in extremely precise scenarios. With more time, these errors could be circumvented, allowing our robot to move all blocks with little trouble.

## References


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

[2] Unknown. (2023) Color filtering in hsv. [Online]. Available: https://cvexplained.wordpress.com/2020/04/28/color-detection-hsv/