Group 7 ArmLab

Nicole Campbell, Varun Aggarwal, Emma Faulhaber, Chinwendu Nwokeabia
{nicoca, varunagg, emmafa, cnwoke}@umich.edu

***Abstract*—The following report is an exploration of the process of programming a ReactorX robotic arm with 5 degrees of freedom (DOF). The arm consists of 7 Dynamixel servo motors. The tasks performed involved block detection and localization using a RealSense L515 RGB and Lidar camera that communicated with a graphical user interface (GUI) through robot operating system (ROS Melodic Morenia). The camera was calibrated using AprilTags on the grid. The end goal was organizing blocks in specific configurations such as sorting blocks by size and color as well as stacking them on top of each other.**

I. Introduction

In this work, the authors developed and implemented algorithms to accomplish autonomous detection of colored wooden blocks in the vicinity of a ReactorX robotic arm using an Intel RealSense L515 LiDAR sensor and manipulation of said blocks to place them in desired configurations. The project consists of two main elements. First, block detection using the OpenCV library. Second, block manipulation using inverse kinematics. All programs for the project were written in the Python programming language. Figure 1 depicts the workspace that includes a ReactorX robotic arm along with a grid and some colored blocks.

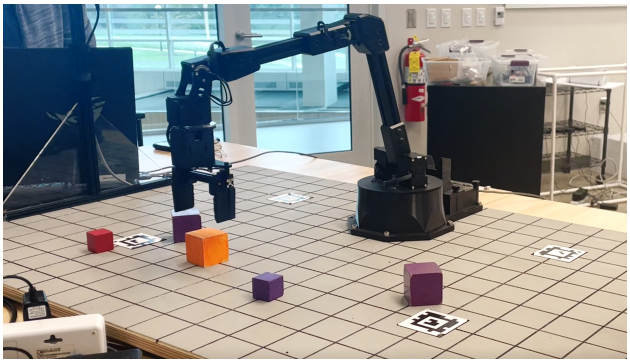

Fig. 1: The ReactorX Arm on the Laboratory Grid with Blocks and AprilTags as during Experimentation

II. Methodology

A. Motor and Camera Calibration

In order to ensure that the robotic arm can reach its target and perform the required tasks even under load, the built in PID controllers for each servo motor were adjusted such that the output angles at the initialized position reached 0 with a tolerance of $\pm 0.7^\circ$ according to the sensor measurements. In order to speed up the response, we increased the proportional gains and in order to eliminate steady-state error, we tuned the integral gain. However, we did not utilize integral gains on the shoulder motors because they were configured in shadow mode.

The camera is also calibrated in two stages, intrinsically and extrinsically. The intrinsic calibration was performed five times using a printed checkerboard. This output four calibration matrices, three of which have been compared to the factory values in Figures 4 and 5. The rectification matrix was a 3x3 identity matrix in all cases, identical to the factory calibration.

The extrinsic matrix was first estimated by taking physical measurements of the camera's location relative to the grid and observing the positive or negative sign of the camera coordinates in different quadrants. The extrinsic calibration was then performed automatically using four AprilTags on the grid at known world coordinates. Figure 6 and 7 show the nominal and auto-calibrated extrinsic matrices, respectively.

The calibration accuracy was verified by checking points in the workspace. The nominal and measured world coordinates are tabulated in Table II for stacks of blocks of 0, 1, 2, 4, and 6 at a point in each quadrant in the workspace.

Equations 1 and 2 are used in conjunction to convert the image coordinates (u,v,d) to the world coordinates (x,y,z). Equation 1 converts the image coordinates to camera coordinates using the depth image value contained in Z_C and the inverse of the intrinsic matrix, K. The intrinsic matrix is the same matrix from the factory testing. Equation 2 converts the camera coordinates to the world coordinates using the inverse of the extrinsic matrix, H. The extrinsic matrix is automatically calculated using iterative Perspective-n-Point (PnP), specifically the OpenCV function `cv2.solvePnP`. To automate this calibration, the points passed into PnP are points automatically detected via the four AprilTags on the workspace.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \end{bmatrix} = Z_C(u,v)K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{bmatrix} = H^{-1} \begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} \tag{2}$$

To verify the calibration system, the physical locations of specific points on the workspace were measured using measuring tape and then compared to the calculated values from the program. To offset distortion effects on the accuracy of the depth values, grid quadrant specific corrections were used. To achieve this, linear relationships between the y-axis and z-axis were calculated for each quadrant of the workspace grid. These relationships were used to add a quadrant-specific linear offset to the depth value.

B. Teach-and-Repeat

Teach-and-Repeat functionally was added to allow the user to guide the robot into a specific position and record the joint angles for the robot to execute again without guidance. This was achieved by adding four new buttons to the user interface: Record Waypoints, Close Gripper, Open Gripper, and Execute Waypoints. Record Waypoints would append the joint angles two a list, Open/Close Gripper would append the command for opening/closing the end-effector, and Execute Waypoints would traverse this list executing both the waypoints and gripper states. The number of times the robot could cycle swap blocks at locations (-100, 225) and (100, 225) through an intermediate location at (250,75) was counted and the joint angles were recorded for one full cycle.

C. Block Detection

To detect the location and properties of the various blocks on the board the arm is to move, we implemented a block detection system using the depth and raw images that the camera uses for the GUI's video feed. The function `blockDetector` in the Camera class in `camera.py` first pulls from color images that are also stored in the VideoFrame class, as well as their corresponding depth image; the color image is then converted to BGR. Masks are automatically (using the location of AprilTags) applied around the robot arm and around the board so that image detection can be limited to just the space where blocks can be placed. Using `cv2.findContours` we applied contours to the blocks based on particular depth threshold values we found when first implementing block detection.

For each contour, the colors within each contour are compared to the dictionary of BGR values we found for each colored block. The block color identified was the color dictionary key with the value that best matched the color within the contour. Block size was also determined by contour size. Block color, the size of the block, and their pixel coordinates were saved for later reference, and the block colors were added to the image to be displayed in the GUI. To address the possibility of the detector categorizing the robot as a block, we also added black to our color dictionary. Additionally, we added an intermediate rest pose for the arm for detection without arm interference. The rate of block detection was also slowed to reduce chances of an inaccurate detection. An example of the Block Detection GUI display can be seen in Figure 11.

D. Forward Kinematics

To find the end effector position in our workspace and display the position in the control station, we calculated end effector forward kinematics using the joint angles for each of the links of the arm. This was done by calculating rigid body transformations that were parameterized by the joint variables. We used the Denavit-Hartenberg (DH) convention to identify the parameters (joint angle, joint offset, link length, and link twist). The DH convention calls for choosing the reference frames of the joints in a way such that the x axes of the current frame would be perpendicular to the z axis of the current and previous frame and intersect the z axis of the current and previous frame. This simplifies the number of needed parameters from six to four. A schematic of the reference frames and links chosen for our calculation for forward kinematics is pictured in Figure 2. See Table VI below for our calculated DH parameters. Each row in the DH table shown represents a homogeneous transformation that follows:

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i}$$

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

where θ_i is the joint angle, α_i is the link twist, a_i is the link length, and d_i is the joint offset for each link i.

The end effector solution was calculated by multiplying each homogeneous transformation for each joint link. The x, y, z, and ϕ values were used to display the end effector in the arm tooltip in the control station GUI.

E. Inverse Kinematics

Given the joint angles, forward kinematics allows us to calculate the pose (position and orientation) of the end effector (tool tip) of our serial robotic arm relative to our time invariant global (world) fixed frame. On the other hand, inverse kinematics (IK) allows us to compute

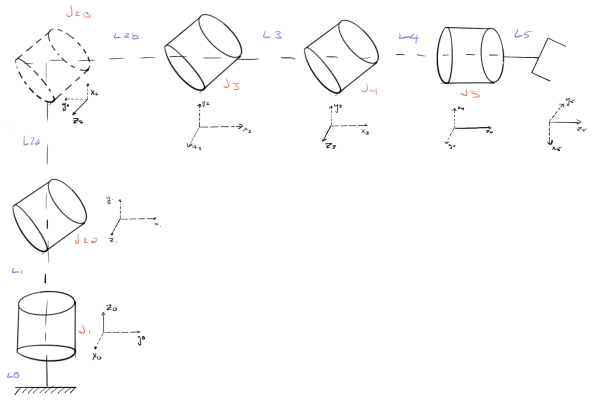

Fig. 2: Schematic of joint links of arm and coordinate system used for forward kinematics

the joint angles required to realize a certain end effector pose relative to the world frame in order to pick and place wooden blocks autonomously.

The IK problem was broken down into 3 segments. Through the individual segment results, on aggregate, we are able to achieve our high level goal of computing the joint angles required to pick a block and place it in the robot's environment autonomously.

The base and world frame lie on the grid and at the geometric center of the cylindrical base of the robot. In order to pick up a block placed either on the grid or stacked on top of another block(s), the arm must be moved such that the end effector points in the general direction of the block. This is segment 1 of the problem. In order to make the end effector point in the general direction of the block, we utilized the block's x and y coordinate in the world frame using equation 3.

$$\theta_0 = -tan^{-1}(x, y) \tag{3}$$

Where θ_0 is the angle the base joint needs to rotate by to make the rest of the arm point towards the block of interest. The negative sign ensures the computed angle for the base motor can directly serve as a command for it. The remainder of the arm is a planar R-R-R arm i.e. a planar revolute-revolute-revolute jointed arm. It was determined that the block should be approached either from the top (vertically, angle of approach = -90 degrees) or from the side (horizontally, angle of approach = 0 degrees). The distance between the block and the shoulder of the arm was calculated using equation 4.

$$\tilde{x} = \sqrt{x^2 + y^2} \tag{4}$$

Additionally, the height (z) offset between the world/base frame and the shoulder frame was corrected using Equation 5 where, z is the z coordinate of the block or the height of the centroid of the top face of the block in the world frame and $0.10391m$ is the height offset. Finally, given the planar nature of our problem, we assigned the z coordinate to complete the planar $\tilde{x}$, $\tilde{y}$ frame associated with the shoulder motors using Equation 6.

$$z = 0.25 - 0.10391 \tag{5}$$

$$\tilde{y} = z \tag{6}$$

The center of our robot's wrist was calculated using equation 8 where, l_3 is the length of the wrist link in meters, r_{03} is the rotation matrix between the shoulder frame as shown in Equation 7, and the end effector (gripper) frame and is a constant depending on the choice of approach angle, theta. In this implementation, it is first attempted to approach the block from the top. If it is physically impossible for the arm to do so, meaning the point is out of the robot's dexterous workspace, the block is approached from the side. Else, the block is unreachable by the arm. Constant lengths for the links associated with the shoulder and elbow motors are defined using the Equations 9 - 11.

$$r_{03} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \tag{7}$$

$$\tilde{xy} = \begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} - l_3 * r_03 * \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{8}$$

$$l_1 = \sqrt{0.2^2 + 0.05^2} \tag{9}$$

$$l_2 = 0.2 \tag{10}$$

$$l_3 = 0.17415 \tag{11}$$

Finally, angles for the elbow θ_2, shoulder θ_1, and wrist θ_3 motors are computed using Equations 12 - 14.

$$\theta_2 = -\frac{cos^{-1}(\tilde{xy}_{11}^2 + \tilde{xy}_{12}^2 - 11^2 - 12^2)}{2 * 11 * 12} \tag{12}$$

$$\theta_1 = tan^{-1}(\tilde{xy}_{12}, \tilde{xy}_{11}) \\ -tan^{-1}(12 * sin(\theta_2), 11 + 12 * cos(\theta_2)) + 1 \tag{13}$$

$$\theta_3 = \theta - (\theta_2 + \theta_1) \tag{14}$$

These equations don't take into account the 14° offset that exists between the shoulder and the elbow joints in the arm. In order to account for the offset, the computed angles were offset as shown in Equations 15 - 16. With a fixed -90 or 0 degree theta value, the wrist angle was computed using equation 14.

$$\theta_2 = \theta_2 - 14 \tag{15}$$

$$\theta_1 = \theta_1 + 14 \tag{16}$$

In the preferred approach of grabbing the block from the top, the wrist rotation motor is also utilized. Using the orientation angle of the block from the block detection algorithm, the angle of rotation for the wrist rotation motor is computed such that it aligns the fingers of the gripper parallel to two opposite faces of the block to enable a robust grab. If the block lies in the first quadrant of the world frame, Equation 17 below was utilized where, θ_4 is the wrist rotation angle and block orientation angle is obtained from the block detection algorithm. If the block lies in the second quadrant of the world frame, Equation 18 is utilized.

$$\theta_4 = -(|\theta_0| + |block\ orientation\ angle|) \tag{17}$$

$$\theta_4 = |\theta_0| + |block\ orientation\ angle| \tag{18}$$

Finally, in order to convert the computed angles into motor rotation commands given the initial (home) position of the robot and the response of the motors to positive and negative rotation commands, Equations 19 - 23 were utilized.

$$m_1 = \theta_0 \tag{19}$$

$$m_2 = -(\theta_1 - 90) \tag{20}$$

$$m_3 = \theta_2 + 90 \tag{21}$$

$$m_4 = \theta_3 \tag{22}$$

$$m_5 = wrist\ rotation \tag{23}$$

Where, m_1 is the rotation command for the base motor (θ_0), m_2 is the rotation command for the shoulder motor (θ_1), m_3 is the rotation command for the elbow motor (θ_2), m_4 is the rotation command for the wrist motor, and m_5 is the rotation command for the wrist rotation motor, which is only applied when approaching the block from above . The inverse kinematics algorithm is implemented in the `state machine.py` program and is implemented in entirety in the `inverse_kinematics` function beginning on line number 341 and ending on line 504. The function consists of nested try-except-else-finally structures. The first try structure tries to compute the joint angles required to grab the block from the top. If it is unable to do so because of a math error due to the block being outside the robot's dexterous workspace, it triggers the nested try structure which tries to compute the joint angles required to grab the block from the side. If it fails as well, it prints a status message to the control GUI saying "point is out of robot's workspace". Figure 3 depicts the arm with all the revolute joints. It specifies each joint by its label and the angle associated with it. Additionally, it specifies the location of the wrist center of the upper planar R-R-R arm. Finally, it highlights the three dimensional x-y-z frame associated with the base joint and the planar $\tilde{x}$, $\tilde{y}$ frame associated with the shoulder joint.

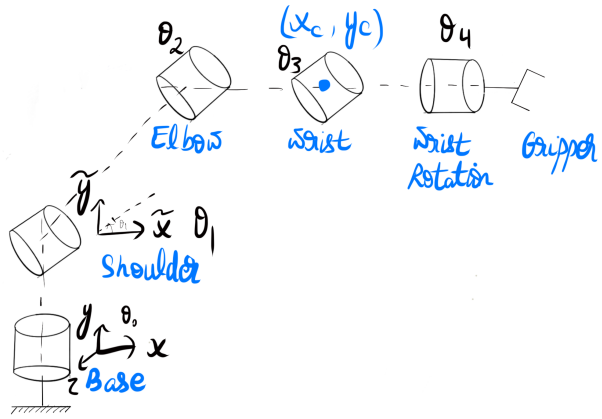

Fig. 3: Inverse Kinematics Parameters

F. Click-to-Grab

The final step needed to control the motion of the arm is a function for the arm to move to a clicked location on the board. This is first implemented in the `click_to_grab` and `click_to_place` state functions in `state_machine.py`. In the click to grab state, the pixel coordinates of the location of the last click, stored in the Camera class, is converted into the world frame. These values are passed into the `inverse_kinematics` function, along with an offset in z to prevent the arm from potentially crashing into a block. The gripper then closes, and the arm moves up an offset amount in z. The same function is used for the click to place command, with a larger z offset values than in the click to grab state.

G. Competition

In order to test the implementation of the arm-camera setup, the following tasks were designed to be achievable at differing levels of difficulty. The easiest levels would typically consist of larger blocks within a smaller color range, while harder levels would involve small and large blocks of all the rainbow colors. The tasks below are described at the level that the team aimed to achieve.

1) Pick 'n Sort!: Event 1 level 3 consisted of starting with 9 randomly colored blocks, possibly stacked, on the positive half-plane of the workspace (above the y-axis center line of the robot), specifically 6 large blocks and

3 small blocks. The robot was tasked to place the large blocks on the right negative half-plane and the small blocks on the left negative half-plane, under 180 seconds.

2) Pick 'n Stack!: Event 2 level 3 started off with the same configuration as Event 1 level 3, but the task was changed to stacking the blocks in the negative half-plane 3 blocks high each.

3) Line 'em Up!: Event 3 consisted of blocks randomly placed on the workspace (possibly stacked up to 4 blocks high). The task was to line up the blocks in a rainbow order in under 600 seconds. Level 1 had only large blocks and level 2 had a mix of small and large blocks.

4) Stack 'em High!: Event 4 had the same starting conditions but instead of lining them up, the task was to stack them in rainbow order.

5) To the Sky!: The Bonus Event consisted of stacking as many large blocks as possible.

III. Results

A. Motor and Camera Calibration

As shown in Table II the maximum error was 10 mm, 13 mm, and 7 mm for x, y, and z coordinates respectively.

Table I shows the PID gains of each motor in the arm. These values achieved $\pm 0.7^{\circ}$ accuracy according to sensor readings from the servos. However, due to the effects of gravity on the arm, it would sag, especially in the shoulder and elbow joints. This compound effect results in the displacement of the arm by up to 6cm in the z-axis. This was corrected by adding a bias in the inverse kinematics when the robot was used to perform specific tasks. Higher gains would have increased the accuracy of the arm in the short term but could potentially damage the motors over time.

TABLE I: Motor PID Parameters

Motors	KP	KI	KD
Waist (1)	640	0	3600
Shoulder (2)	800	0	0
Elbow (3)	800	200	0
Wrist Angle (4)	800	200	0
Wrist Rotation (5)	640	0	3600

Figure 4 shows the manually derived calibration values as an average of the 5 calibrations done using a printed checkerboard. Figure 5 are the factory values for a similar calibration done with higher quality equipment.

$$\begin{bmatrix} 936.09 & 0 & 640.03 \\ 0 & 934.30 & 378.79 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) Intrinsic Matrix, K (mm)

$$\begin{bmatrix} 0.1553 & -0.2665 & 0.0099 & 0.0018 & 0 \end{bmatrix}$$

(b) Distortion Matrix (D)

$$\begin{bmatrix} 950.79 & 0 & 642.28 & 0 \\ 0 & 961.45 & 384.77 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(c) Projection Matrix, P (mm)

Fig. 4: Manual Calibration Matrices

$$\begin{bmatrix} 907.90 & 0 & 641.05 \\ 0 & 908.36 & 354.88 \\ 0 & 0 & 1 \end{bmatrix}$$

(a) Intrinsic Matrix, K (mm)

$$\begin{bmatrix} 0.1556 & -0.4935 & -0.0009 & 0.0008 & 0.4401 \end{bmatrix}$$

(b) Distortion Matrix (D)

$$\begin{bmatrix} 907.90 & 0 & 641.05 & 0 \\ 0 & 908.36 & 354.88 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

(c) Projection Matrix, P (mm)

Fig. 5: Factory Calibration Matrices

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 188 \\ 0 & 0 & -1 & 970 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 6: Nominal Extrinsic Matrix (mm)

$$\begin{bmatrix} 0.999 & 0.004 & -0.006 & 29.981 \\ 0.005 & -0.999 & 0.026 & 183.236 \\ -0.006 & -0.026 & -0.999 & 973.334 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 7: Example of Automatically Calibrated Extrinsic Matrix (mm)

The calibration was tested by stacking blocks at different known locations on the board. The final calibration values resulted in the results shown in Table II and included in the appendix for readability.

TABLE II: Measured v.s. Nominal Coordinates from Calibration

# Blocks (Nominal Z)	(x, y, z) Coordinates in mm			
(x, y) Location	0, 175	-300, -75	300, -75	300, 325
0 (0±2mm)	-1,175,1	-308,-82,-1	308,-82,-1	306,326,0
1 (38±2mm)	-1,178,40	-309,-86,36	308,-82,-1	305,326,36
2 (77±2mm)	-1,178,75	-310,-85,73	308,-87,71	309,327,73
4 (153±2mm)	0,174,155	-309,-86,153	308,-87,148	306,323,152
6 (230±2mm)	-1,172,228	-308,-88,227	308,-82,223	307,325,225

It was observed that after calibration, the world coordinates displayed in the GUI when hovering over the blocks at known coordinates remained within $\pm 11mm$ accuracy in the (x, y) plane. Depth was even more accurate within $\pm 5mm$. It was also found that the blocks were not perfect cubes and therefore the nominal depth should be considered to vary within $\pm 2mm$.

B. Teach-and-Repeat

The teach-and-repeat function was used to cycle switching the positions of two blocks using an intermediate point to place the block, as only one could be reliably picked up. The robot was able to cycle through this motion autonomously 5 times before errors in its positioning became so large that it missed picking up a block and thus failed the cycle. A graph of the joint angles for each motor during one cycle of the taught motion can be seen in Figure 8. For readability, refer to Figure B.1 in the appendix for the same plot.

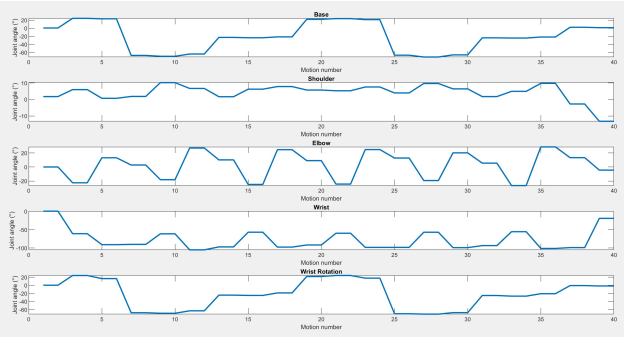

Fig. 8: Joint Angles for One Cycle of Teach-and-Repeat

The main contributor to the errors in arm motion was the sag of the arm. As it reached down to grab a block it had been taught to approach the table too closely and with the added sag would hit the table and impact the motion of the arm. When placing the block, it did so from a height that caused the block to fall and be displaced slightly from the desired position. To improve, the teaching could have accounted for these biases or the repeat function could have some built in. However, these biases were accounted for in the next phases of the code to allow for reliable block manipulation. Additionally, continuous block detection allowed for blocks that had been missed while grabbing, to be detected again and allow the robot to try to grab them again.

C. Block Detection

We calculated and stored the coordinates of the contours for a series of blocks for several trials to assess the robustness of our block detector. We also found the actual image coordinates of each of the blocks using the control station GUI; these image coordinates acted as our 'ground truth' to compare the contour coordinates from our block detector. Figure 9 below, is an image of the block detector's contours juxtaposed with the actual blocks. After data collection, we plotted the block detections and ground truth blocks based on the values collected. See Figure 10 to see the measurement of the block detections compared with the actual block coordinates (without orientation angles taken into account). As seen in Figures 9 and 10, the block detection was a bit noisy and inaccurate, with the contours being often not large enough to account for the whole block. In addition, block detection did have some trouble with identifying smaller blocks on top of larger blocks. One way we could have addressed this is by finding a more robust means of classifying larger and smaller blocks, possibly segmenting block detection on the basis of depth value of the block, and by implementing some sort of filtering technique to reduce the noise and jumpiness of detections found. A larger image of Figure 10 can be found in Figure D.3 in Appendix D.

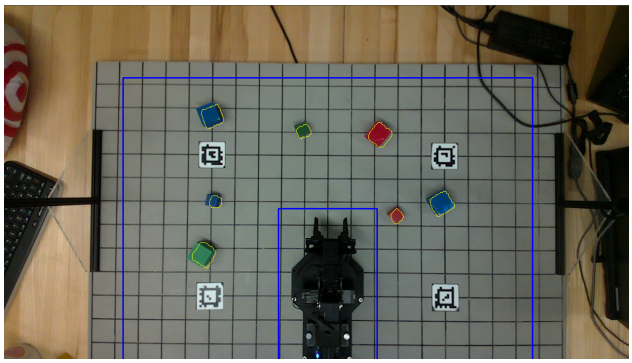

Fig. 9: Image of sample block detection overlayed with Block Detector contours in yellow

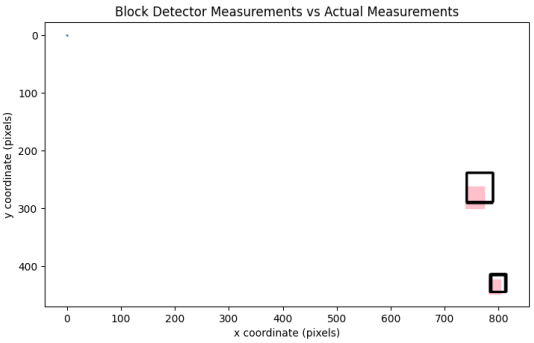

Fig. 10: Uncertainty of Block Detection vs Ground Truth

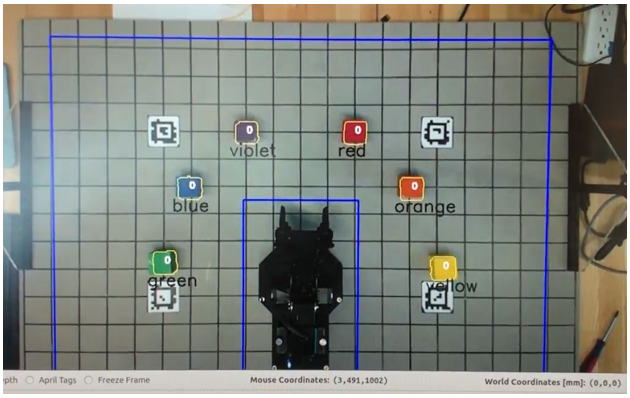

Fig. 11: Example image of Block Detector GUI display

D. Forward Kinematics

For our calculation of forward kinematics, we found the necessary values to complete each joint's row of the DH table. The DH table we ultimately used for forward kinematics can be seen in Table VI. Verifying forward kinematics included implementation of FK for the end-effector's tooltip and comparing the end-effector orientation based on FK with the actual location of the end-effector in the workspace. See Table III for the predicted end-effector positions (x, y, z) and (θ, ϕ) for the arm in its initial (home) configuration in the control station GUI. Table IV shows the predicted and actual (x, y, z) coordinates of the end effector in various poses and Table V shows the percent error between real world and calculated end-effector pose. Based on the error calculations in Table V, the largest error values correspond to the measurements along the z-axis. This may be due to the effects of gravity on the arm which caused a moderate level of sag on the arm.

TABLE III: End Effector Location

X	0.07 mm
Y	386.64 mm
Z	280.88 mm
Phi	-176.23 rad
Theta	0
Psi	0

TABLE IV: FK Pose vs Real World Pose

Trial	x_{FK}	y_{FK}	z_{FK}	x_{rw}	y_{rw}	z_{rw}
1	0	380.96	303.98	0	370	260
2	213.7	370.1	154.1	205	320	115
3	175.5	304	157.9	175	310	125
4	182.1	315.4	213	180	275	195
5	315.4	182.1	213	295	180	190

TABLE V: Percent error: FK vs Real World Pose

Trial	x_{error}	y_{error}	z_{error}
1	$\sim$	2.874	14.468
2	4.071	15.536	25.373
3	0.285	1.974	20.836
4	1.153	12.809	8.450
5	6.468	1.152	10.798

TABLE VI: DH Table

Joint	Link	a_i	$\alpha_i(^\circ)$	$d_i(m)$	$\theta_i(^\circ)$
1	1	0	90	0.10391	$90 + \theta_1$
2	2_y	0.2	0	0	$90 + \theta_2$
2	2_x	0.05	0	0	-90
3	3	0.2	0	0	$0 + \theta_3$
4	4	0	90	0	$90 + \theta_4$
5	5	0	0	0.17415	$180 + \theta_5$

IV. Discussion

A. Camera Calibration

The lab calibration matrices for the camera and projection remained within a $\pm 10\%$ difference from the factory calibration. However, the distortion matrix varied massively, up to 1219%, and was often not even within the same decimal range. Factory calibrations would have been done in optimal lighting using a much more textured target that didn't suffer from wear like the one in the lab had. This would have caused differences in all the intrinsic matrices, especially distortion as this calibration is highly sensitive to any variations in depth in the scene. In this case, there were variations induced by the camera not being aligned perfectly parallel to the

grid, as the stand was angled inside the machined slots on the side, and the grid itself being on an angle on the table. Therefore, after trying the different values, it was determined that the factory values were preferable and were applied in the final code.

It can be observed that the manual and automatic extrinsic matrices shown in Figures 6 and 7 are very close in most values, but the auto-calibrated matrix has more floating point precision. Key differences are the translation in the x-axis ($\pm 29mm$) and small rotation values which are due to the camera lens placement relative to the workspace. It is not sufficient to rely on the nominal extrinsic matrix because with any perturbation of the camera or workspace, it will be incorrect and will need to be recalculated by hand.

B. Competition

For the ArmLab Competition, the robot was able to successfully perform level 3 for both Events 1 and 2 completely autonomously. Events 3 and 4 were also completed, but required the use of Click-and-Grab functionality with only large blocks (level 1). For the Bonus Event, the robot was able to stack up to 6 blocks. However, while performing the exiting maneuver, it would not pull back far enough and as a result, knocked over the topmost block.

While the nominal extrinsic matrix provided a decent estimate and good orientation of the axes, any perturbation to the camera or offset that wasn't accounted for using the general measurements with a ruler and string would cause the calibration to be offset. Therefore, an automatic calibration procedure was designed wherein the user would click a calibration button in the GUI and the AprilTags at known coordinates in the world frame would be used to calibrate the camera to world frame values automatically. An example of an automatically calibrated extrinsic matrix has been included below in Figure 7.

1) Improvements: One of the key problems the autonomous system faced was picking up the blocks in incorrect orientations or not perfectly over the center of mass; even a slight error in these caused the stack of blocks to be unstable when stacked. To correct for this, a potential intermediate position could be implemented where it would be easier for the robot to pick up the block in the correct orientation.

Another key problem was not exiting cleanly while stacking after placing 5+ blocks. This was due to the robot running out of dexterous workspace since it was originally programmed to go up in the z-axis 100 mm after placing a block. One way to solve this would be to provide an exiting way point to make sure the stack was completely cleared before the robot executed any other waypoint.

V. Conclusion

The overall goal of the robotic system was to be able to leverage the RGB camera and the LiDAR depth data to visualize and manipulate a workspace with colored wooden blocks using a 5 DOF robotic arm. Camera calibration properties such as intrinsic and extrinsic matrices in conjunction with tools from the OpenCV library such as the `solvePnP` for camera calibration and `findContours`, moments, rectangle for block detection allowed for processing of the visual data. The calibration procedure with an added distortion correction for the depth values resulted in all x, y, and z coordinates in the world frame having less than 10 mm, 13 mm, and 7 mm error for x, y, and z respectively as shown in Table II. Using OpenCV, information from the blocks such as color, centroid, and orientation data for the blocks were collected for later use to manipulate the blocks.

With all the necessary data for each block in the workspace such as location and orientation, the robotic arm was able to grab and place the blocks. The kinematics of the arm were achieved with forward kinematics which found the end effector position using joint angles of the arm, whereas the inverse kinematics calculated the joint angles given an end effector pose. To automate movement of the robot, inverse kinematics was used.

For future work, a more robust block detection algorithm would allow the system to save cycle time while maneuvering blocks, since the current algorithm requires the system to move the arm to a position where block detection is not processed to avoid noise. Additionally, a more robust controller for the motors will allow the arm to reach the desired pose accurately and precisely whilst accounting for the sag autonomously, thereby eliminating the need for manual corrections in grab and place operations.

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] D. Eggert, A. Lorusso, and R. Fisher, "Estimating 3-D rigid body transformations: a comparison of four major algorithms," *Machine Vision and Applications*, vol. 9, no. 5, pp. 272–290, Mar. 1997. [Online]. Available: https://doi.org/10.1007/s001380050048

[3] "Color spaces in OpenCV (C++/Python) | LearnOpenCV," May 2017. [Online]. Available: https://learnopencv.com/color-spaces-in-opencv-cpp-python/

APPENDIX A
CAMERA CALIBRATION TEST DATA

TABLE I: Measured v.s. Nominal Coordinates from Calibration

# Blocks (Nominal Z)	(x, y, z) Coordinates			
(x, y) Location	0, 175	-300, -75	300, -75	300, 325
0 (0±2mm)	-1,175,1	-308,-82,-1	308,-82,-1	306,326,0
1 (38±2mm)	-1,178,40	-309,-86,36	308,-82,-1	305,326,36
2 (77±2mm)	-1,178,75	-310,-85,73	308,-87,71	309,327,73
4 (153±2mm)	0,174,155	-309,-86,153	308,-87,148	306,323,152
6 (230±2mm)	-1,172,228	-308,-88,227	308,-82,223	307,325,225

APPENDIX B
GRAPH OF MOTOR OUTPUTS DURING TEACH-AND-REPEAT

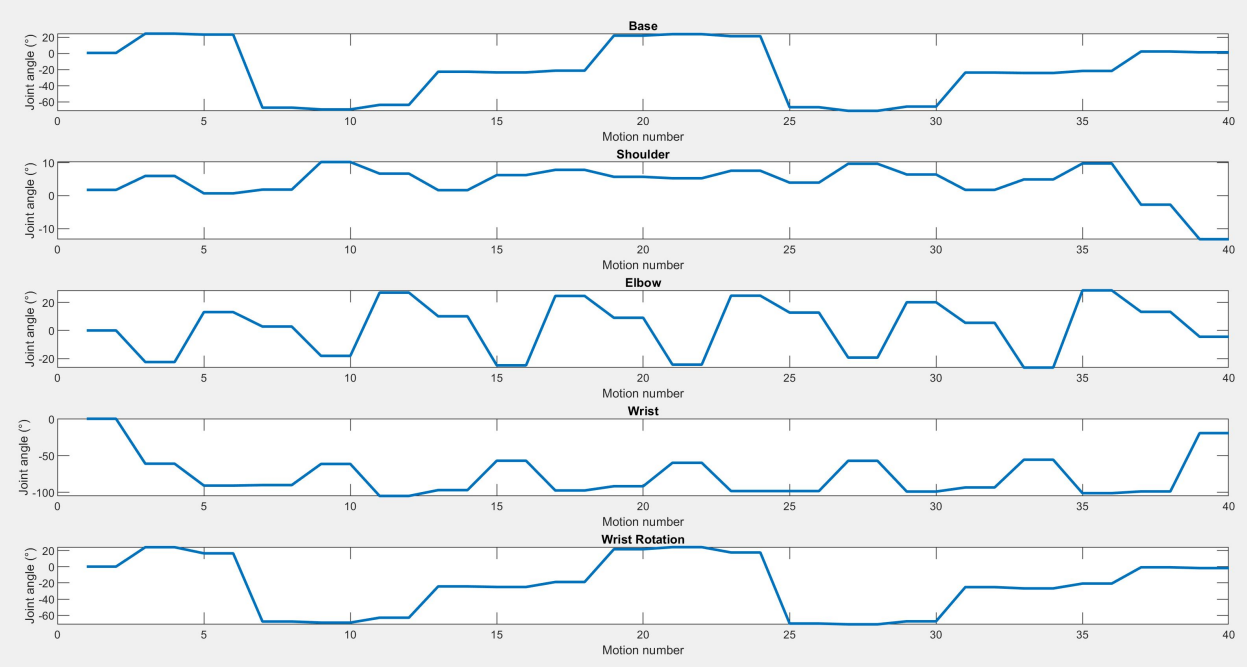

Fig. B.1: Joint Angles for One Cycle of Teach-and-Repeat

APPENDIX C
FORWARD KINEMATICS DH FRAMES

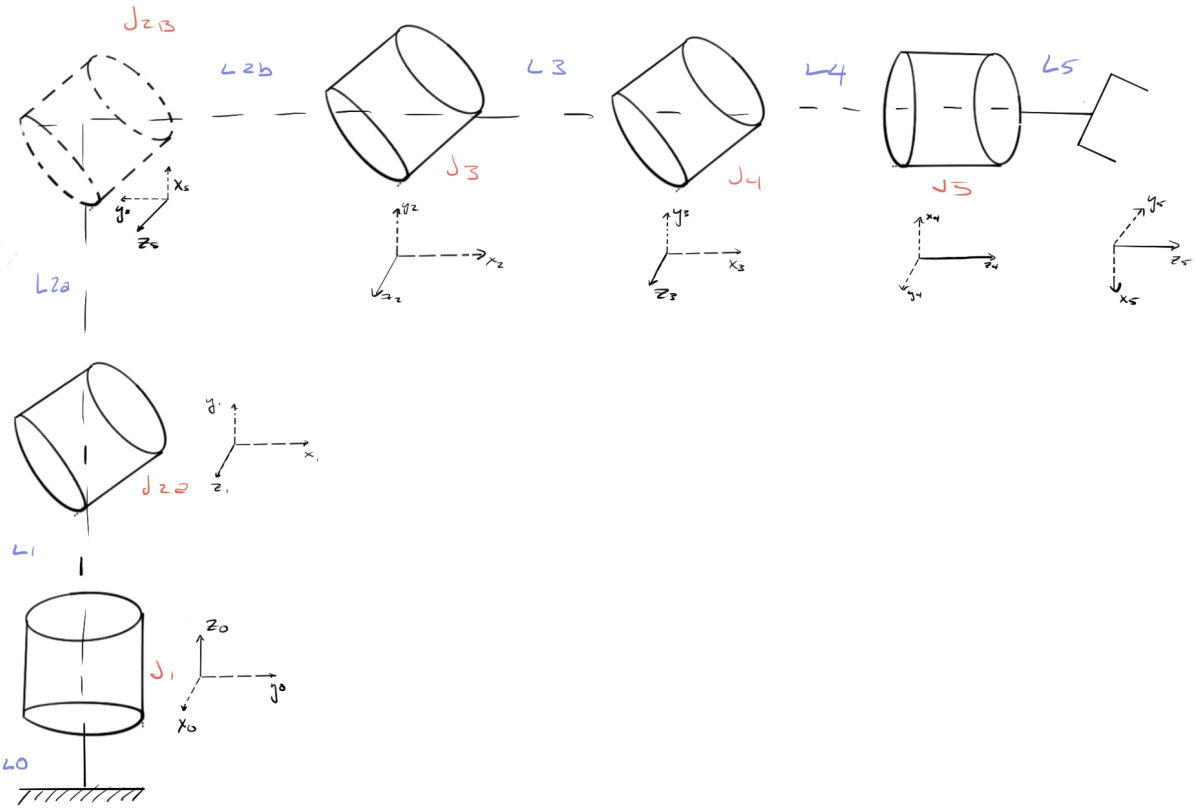

Fig. C.2: Schematic of joint links of arm and coordinate system used for forward kinematics

APPENDIX D
BLOCK DETECTION UNCERTAINTY

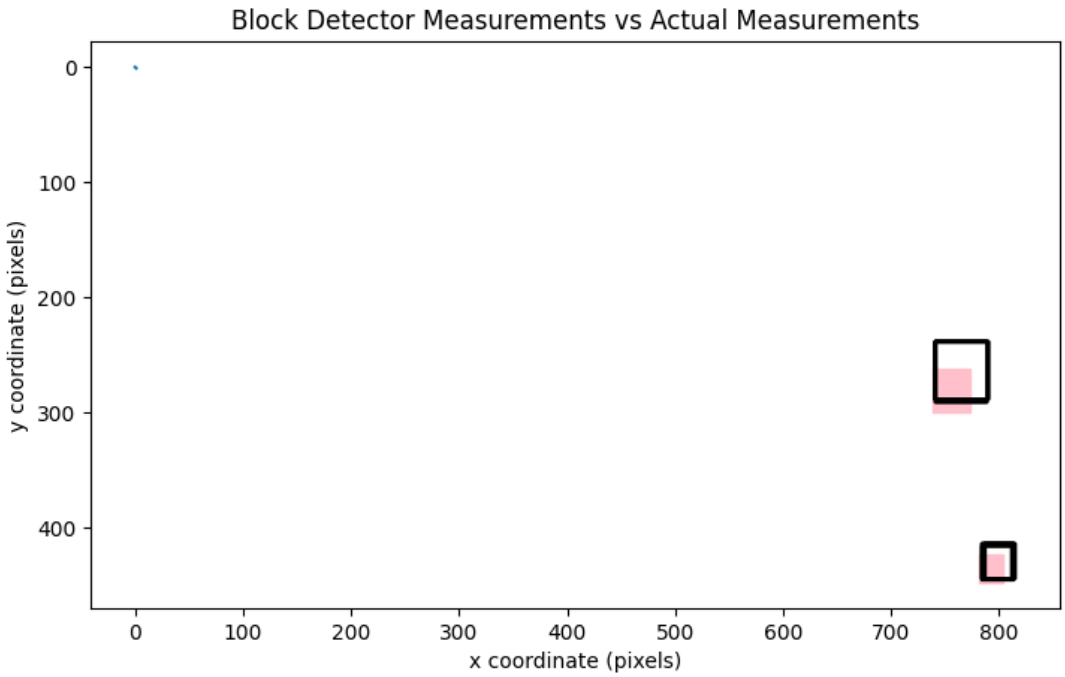

Fig. D.3: Uncertainty of block detection vs ground truth