ROB 550 Armlab Report - RobotArm

Aaron Tran*, Robert Frei*, and Shashank Verma+

Abstract— **This report details the completion of the Armlab portion of ROB 550 in Winter 2023 by group six for the PM section. In this project, a realsense LiDAR camera was provided for visual sensing of wooden blocks, and a 5 degree-of-freedom (DOF) robotic arm was provided to move the blocks to predetermined positions and orientations. The software development detailed here includes camera calibration using fiducial markers and conversion from RGB position and depth to cartesian locations of wooden blocks, as well as the forward and inverse kinematics of the robotic arm to move those blocks to predefined positions for several challenges. Overall, this project was successful in visually detecting blocks, grasping them, and moving them to new locations as a function of their size and color, and achieved second place in the lab competition.**

I. INTRODUCTION

A robotic arm manipulator is a type of robotic system that is designed to mimic the movements of a human arm. It is a highly versatile and flexible tool that can be used for a wide range of applications, from manufacturing and assembly to medical procedures and even space exploration.

The robotic arm manipulator typically consists of a series of linked segments, each of which is driven by an actuator or motor. The segments are connected by joints that allow the arm to move in a variety of directions and orientations. The end of the arm is equipped with a tool or gripper that can be used to manipulate objects, such as picking up and moving items on a production line or performing a surgical procedure. There are many different types of robotic arm manipulators available, ranging from simple, two-jointed systems to more complex systems with six or more joints. The level of complexity of the system will depend on the intended application, with more complex systems typically offering greater flexibility and precision.

The development of robotic arm manipulators has been a major focus of research in the field of robotics for many years, with new advancements being made on a regular basis. These advancements include improvements in control systems, materials, and sensors, as well as the development of new tools and end-effectors.

This paper reports the work done in the Armlab for ROB 550 Winter 2023. The ReactorX 200 robot arm was used, which is a 5 DOF manipulator with 7 dynamixel servo motors with 3D printed custom end effectors. An Intel RealSense LiDAR Camera L515 sensor, a solid state LiDAR depth camera, was used for computer vision sensing. The realsense provided both depth data and an RGB image. The first objective of this project was to accurately detect small and large blocks in the workspace and distinguish them based on color, size, and shape. The second objective was to autonomously lift and place them in predefined postions and orientations. The dimensions of the small and big rectangular blocks were 25 and 35 mm, respectively. The workspace also contained several non-cubic distractor blocks, which were avoided while performing predefined tasks.

The contents of the paper are as follows. Section II presents the algorithmic development of the vision and kinematic systems, Section III describes the tests and competition performed to evaluate those algorithms, Section IV discusses the robot performance from the results section, and Section V concludes the paper.

II. METHODOLOGY

This section covers the algorithm derivation and implementation for camera calibration and euclidean position conversion, block feature extraction from depth and RGB images, forward arm kinematics, inverse arm kinematics, and our state machine.

A. Workspace Reconstruction

We detect blocks with an RGB-D camera but manipulate the robot arm in the world frame. Occasionally, it is beneficial to work with world coordinates, but other times pixel coordinates are more convenient. The following describes our methodology for converting between the two.

1) Intrinsic Matrix: The intrinsic matrix is a homegenous tranformation from camera frame to pixel frame. We used a pinhole camera model [Fig. 1] which gives us an intrinsic matrix of the following form:

$$\begin{bmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

f is the focal length. x_0 and y_0 are optical axis offsets for the x any y axis. We obtained an intrinsic matrix using a calibration tool from ROS, but ended up using the factory calibrated intrinsic matrix.

2) Extrinsic Matrix: The extrinsic matrix is a homogenous transformation from world frame to camera frame. Our method utilizes fiducials to solve the Point-n-Perspective (PnP) problem. Specifically, four AprilTags [1] were placed on a flat surface and detected automatically using Robot Operating System (ROS) to detertmine their relative position to the camera. PnP describes the problem of relating our 2D

*Aaron Tran and Robert Frei are with the Department of Robotics, University of Michigan, Ann Arbor, MI 48109, USA. E-mail: {aartran, freir}@umich.edu.

+Shashank Verma is with the Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI 48109, USA. E-mail: shaaero@umich.edu.

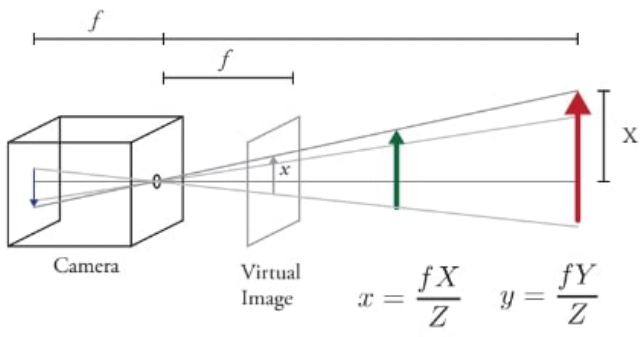

Fig. 1: Pinhole camera model. f describes the focal length of the camera, x and y describes the horizontal and vertical in the image frame, and X, Y, and Z describe the horizontal and vertical coordinates in the camera frame.

(pixel frame) and 3D (world frame) point correspondences through rotation and translation. With the intrinsic matrix, we can change this problem to look for correspondences between points in the camera frame and points in the world frame. We use four AprilTags to obtain the four correspondences. The known size of the AprilTags determine their relative position to the camera from pixel readings. cv2.solvePnP() solves for the rotation and translation matrices, which we compile into the extrinsic matrix.

We considered using an extrinsic matrix using hand measurements and a SVD-based method described in [2]. The hand measurement involves measuring the frame of the camera in world coordinates and inverting it. The SVD-based method involves selecting 18 points correspondences and running them through the calculations described in the paper. Our final implementation uses the apriltag method.

3) Pixel to World: The following formulation converts from pixel coordinates to world coordinates:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c(u,v) \times IntrinsicMatrix^{-1} \times \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad (2)$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = ExtrinsicMatrix^{-1} \times \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (3)$$

where $Z_c(u,v)$ is the depth sensor reading at pixel coordinates (u,v). Subscripts of c and w indicate coordinates in the camera and world frame.

4) World to Pixel: : The following formulation converts from world coordinates to pixel coordinates:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = EntrinsicMatrix \times \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \quad (4)$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = 1/Z_c \times IntrinsicMatrix \times \begin{bmatrix} \mathbf{I} & 0 \end{bmatrix} \times \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \quad (5)$$

B. Block Detection

Block detection consists of two steps: segmentation and data extraction. Segmentation refers to the individual extraction of each distinct object in the image. We then extract information from the segmented images.

1) Segmentation: Our block detection segments blocks from the recorded image using height. Our algorithm maps the world Z height to corresponding pixel locations and thresholds the image using a minimum Z height. This provides us with a binary image, where any block of sufficient height is white and everything else is black. To treat noise in the image, we apply dilation followed by erosion. Dilation is similar to an OR operation and will set a pixel value to 1 if *any* pixels in the neighboring area are 1, setting it to 0 otherwise. Erosion is similar to an AND operation and will set a pixel value to 1 only if *every* pixel in a specified neighouring area is also 1, setting it to 0 otherwise. These steps allow the vision processing system to remove white noise and then expand noisless areas back to their original size. We accomplished this using cv2.morphology() with a 4x4 kernel and op = cv2.MORPH_CLOSE. We apply a mask to limit detection to only areas of interest, limiting sources of noise. This is shown in Fig. 4. To identify the top of a stack without allowing perspective effects (seeing the side of the tower) to affect the segmentation, each segmented object goes through a redetection algorithm. Redection applies a new segmentation that masks out all but the initially identified object and thresholds for 5 mm below the max identified height of the initial segmentation. The final object is identified using a contour, found by cv2.findContour().

To account for depth camera issues, our implementation applies an offset to the original depth frame image as part of the calibration procedure. After calculating intrinsics and extrinsics, we obtain a depth frame image shown in Fig. 2. This depth frame has been processed to show the height values instead and has been colorized to make it easier to see differences. The z values range from around -30 at the bottom of the board to 10 at the top of the board. We set the offset equal to this entire frame and is added to every raw depth frame. With this we were able to obtain a new processed depth frame, shown in Fig. 3

2) Data Extraction: We are interested in the position, orientation, color, and block type of each block. We find position by finding the contour's centroid, which can be calculated from its moments. The moment equation is:

$$M_{ij} = \Sigma_x \Sigma_y x^i y^j I(x,y) \quad (6)$$

where (x,y) is a pixel coordinate and $I(x,y)$ is the value of the pixel at the coordinate (1 or 0). The centroid is calculated using:

$$(C_x, C_y) = (\frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}}) \quad (7)$$

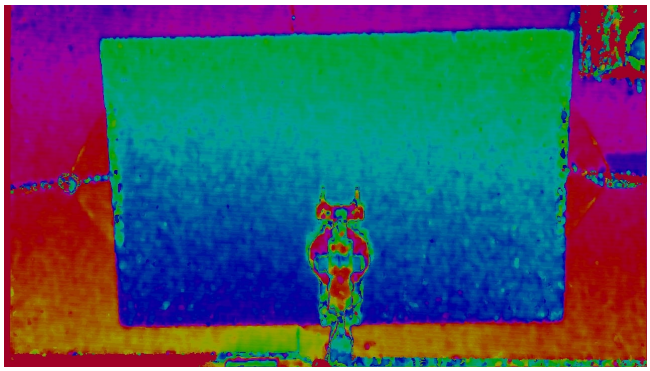

Fig. 2: Colorized depth frame before being zeroed

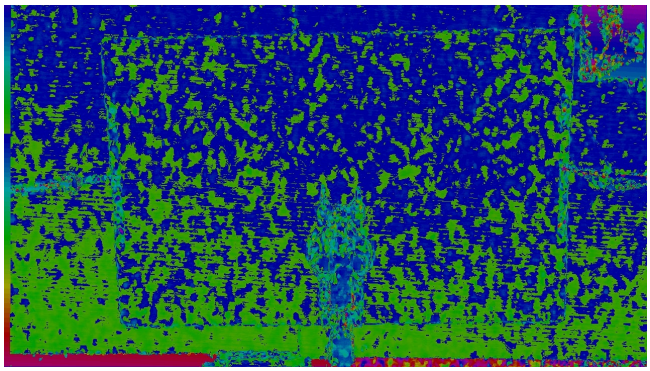

Fig. 3: Colorized depth frame after being zeroed

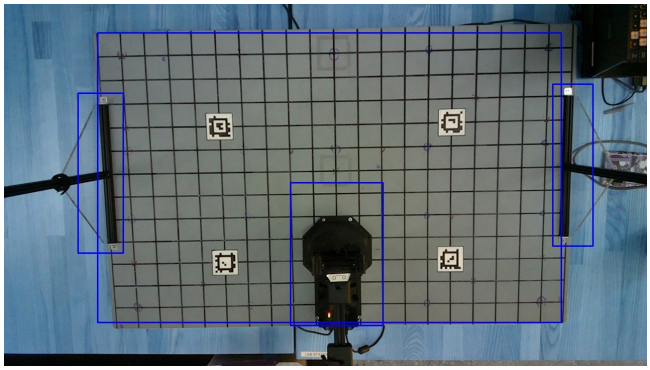

Fig. 4: Mask applied to image. The largest rectangle provides a mask for the area of interest. All of the three other masks are overlayed on this to indicate areas of disinterest. cv2.findContour() only considers areas of interest when constructing contours.

Passing this through equation (2) returns the position of the block in world coordinates where $Zc(u, v)$ is chosen to be the minimum depth value across the contour.

We assume that the camera's z-axis is close to parallel with the world z-axis, allowing us to say that each block's orientation in the image frame is identical as that in the world frame. We also assume that each block's rotation is purely in z. cv2.minAreaRect() returns a rectangle around the contour of minimum area, giving us the orientation, where a rectangle with sides parallel to the image borders has an orientation of $0°$.

The camera takes images in the RGB colorspace, but RGB is lighting sensitive. We instead convert the colorspace to LAB, where L measures the lightness and A and B encapsulate the color information. To determine block colors, we average the A and B values across the pixels encapsulated by the contour and calculate the euclidean norm between those values and some prespecified values for red, orange, green, yellow, blue, and violet. We assign the color with the smallest euclidean norm with the mean to the block.

The algorithm for block shape identification also uses euclidean norms to prespecified values. cv2.minAreaRect() returns the width and height of the block as well. We take the minimum of these and rename that as width, leaving the remaining value to be the height. We compare the norm between this and the width and height of the blocks. Smallest norm determines assignment. Additionally, using the difference in the two side lengths, the robot will always align the end effector to pick up the blocks along the shorter side for non-cubic shapes.

C. Homography

Homography or the projective transform, is an affine transform that allows us to move from one camera view to another. This allows us to account for perspective distortion. We calculated a homography matrix using cv2.findHomography() to calculate a homography matrix and cv2.warpPerspective() to apply it to the video frame, but it is not used as a part of block detection. We use it to clean up the image by transforming to a view that only includes the grid. Fig. 7 shows the video frame after the homography transform has been applied. To accomplish this, we gave the position of the four corners of the grid in pixel coordinates and specified correspondences to the four corners of image frame.

The LiDAR is not necessarily aligned with the camera axis. With large discrepancies between the two, a homography matrix may be necessary; however, we found that our frames matched well. We did not compute a homography transform to make depth and RGB images come from the same perspective.

D. Forward Kinematics

Forward kinematics (FK) is used to determine the position end effector (X), where $X = [x, y, z, \theta, \psi, \phi]$ in the workspace using only configuration state (Q), where $Q = [q_1, q_2, q_3, q_4, q_5]$. The configuration states are the joint angles of the robot arm corresponding to the 5 DOF. We solved the FK probem using the Denavit-Hartenberg method [3].

In the DH method, we first identify different links and joints and numbered them. Next, frames are assigned to all the joints based on DH conventions as shown in Figure 5.

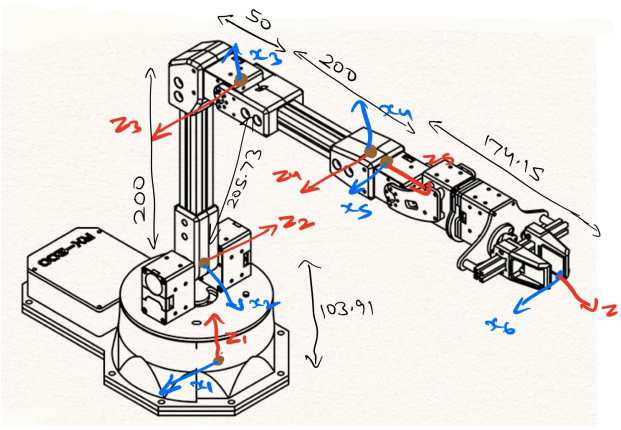

Fig. 5: Schematics of the robot arm. Frames are marked at the joint locations using DH convention

	θ	d	a	α
Link 1	q_1	103.91	0	90°
Link 2	q_2	0	205.73	0
Link 3	q_3	0	200	0
Link 4	q_4	0	0	90°
Link 5	q_5	174.15	0	0

TABLE I: DH parameters

Forward kinematics map is represented as a vector function $X = f(Q)$. We use DH algorithm to determine the function f. Shoulder and elbow frames have a angular offset of 14°, and the same is included in Table I. The homogeneous transformation (H) matrix is calculated by multiplying the homogeneous transformation matrix of each link

$$H = A_1(q_1)...A_n(q_n) = \begin{bmatrix} R_n^0 & O_n^0 \\ 0 & 1 \end{bmatrix}, \tag{8}$$

where R_n^0 is the rotation matrix of the end effector wrt world frame. O_n^0 represent the location of the end effector in world frame. The Euler angles of the end effector roll ϕ, pitch θ, and yaw ψ are calculated using the rotation amtrix R_n^0.

The zero of the shoulder arm axis and the zero of the forward kinematics are at an angle of 90°. The shoulder and elbow joint has an offset of 14°. To account for this in Table I, Q is modified as

$$q_1 = q_1 + 90^\circ \tag{9}$$
$$q_2 = -q_2 + 14^\circ \tag{10}$$
$$q_3 = q_3 - 14^\circ \tag{11}$$
$$q_4 = q_4 + 90^\circ \tag{12}$$
$$q_5 = q_5 \tag{13}$$

E. Inverse Kinematics

Inverse kinematics (IK) is used to get the joint positions i.e., configuration state (Q) using end effector position and orientation. Geometric method is use for inverse kinematics along with the kinematic decoupling, where 5 DOF robotic arm is split into two parts, namely $i)$ base to wrist joint and $ii)$ wrist to end effector.

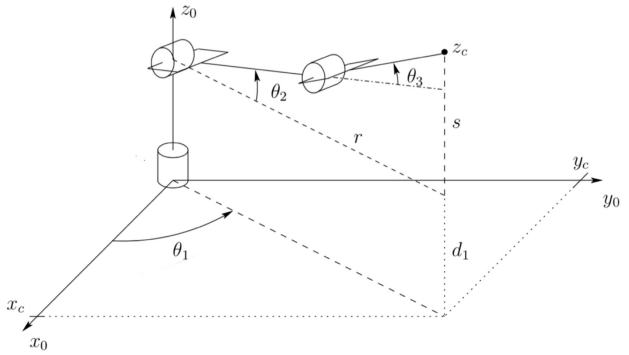

Fig. 6: Inverse kinematics for base, shoulder, and elbow joint

Configuration of the joint angles from the base to elbow is shown in Figure 6. Using geometry, the joint angles q_1, q_2, and q_3 is obtained using the end effector location (x, y, z), which is given in the world frame. The orientation of block is obtained using the block detection using the *cv2.MinAreaRect* function. This give us the pose of the end effector i.e., $(x, y, z, \phi, \theta, \psi)$. Two different configuration of the wrist is use, namely, 'flat' and 'down.' In the flat configuration, the gripper is always parallel to the workspace plane where as in the down configuration, the gripper is always perpendicular to the workspace plane.

When wrist configuration is flat, the location of the wrist joint is given by

$$R_n^0 = R(atan2(y, x), 90^\circ, 0), \tag{14}$$
$$O_c = O - l_6 R \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{15}$$

where, R function gives rotation matrix from euler angles using 'ZYZ' configuration and $l_6 = 174.15$. When wrist configuration is down, the location of wrist joint is given by

$$R_n^0 = R(0, 180^\circ, 0), \tag{16}$$
$$O_c = O + l_6 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{17}$$

where, $O_c = [x_c \ y_c \ z_c]^T$ is the location of wrist in the world frame and $O = [x \ y \ z]^T$.

$$\theta_1 = atan2(y_c, x_c) \tag{18}$$
$$q_1 = \theta_1 - 90^\circ \tag{19}$$
$$\theta_3 = acos((x_c^2 + y_c^2 + (z_c - d_1)^2 - l_1^2 - l_2^2)/(2l_1 l_2) \tag{20}$$
$$q_3 = \theta_3 + 76^\circ \tag{21}$$
$$q_2 = 76^\circ - \theta_2 \tag{22}$$

where,

$$\theta_2 = atan2((z_c - d_1), \sqrt{x_c^2 + y_c^2}) - atan2(l_2 \sin\theta_3, l_1 + l_2 \cos\theta_3) \quad (23)$$

where, $d_1 = 103.91, l_1 = 205.73, l_2 = 200$. Rotation matrix R_3^0 is generated using FK and using rotation matrix property $R_n^3 = (R_3^0)^{-1} R_n^0$. Wrist joint angles q_4, q_5 are equivalent to 'ZYZ' Euler angles which can be obtained from R_n^3. THe above equations can be found in the 'kinematics.py' file under 'IK_geometric_event_1' function.

F. State Machine and Arm Controls

To execute and order programmable commands in real time, our robot runs a state machine script. In that script, an infinite loop checks for next_state, sets current_state to that value, and runs the corresponding function. These functions include executing waypoint trajectories, calibrating the camera, running competition rounds, and others. At the conclusion of each state function, next_state is set so that upon re-running the loop, the next task may be performed. In the gui interface, when buttons are clicked to perform tasks, a script checks to see if the state is idle, and if so sets the next state to the desired task.

All motor control is performed by building a list of desired motor positions and a corresponding gripper state list. The desired arm configuration at any one point is called a waypoint. After compiling the list, we call StateMachine.execute(), which loops through the list and calls rxarm.set_positions(). This script assigns a trapezoidal kinematic profile to each joint between the current and desired angles with set acceleration and travel times. To determine when a waypoint is reached, we check on every loop of the state machine whether the arm has moved within sufficient error to the desired position or the motion duration has exceeded a specified timeout period. In addition, to account for the weight of the arm in further reaching configurations, a linear correction offset is added in z to the desired endpoint position as a function of distance from the base of the robot. In the absence of force feedback, this correction was able to maintain a near-constant z-height regardless of block position on the board.

III. MOTION PLANNING

The only functionally significant action we perform with the arm is some form of pick and place. Accordingly, our motion planning only pertains to pick and place operations. Our planning takes a relatively simplistic approach that can be broken down into 8 steps: approach, descend, grasp, ascend, approach, descend, place, ascend. Each block is assgned a pick frame and a place frame. The arm approaches the place frame from 150mm above. This is higher than a four block stack, so it should prevent the arm from knocking over towers while reaching this position. It then descends onto the block, grasps it, and ascends back up to the approach position. From there it moves to the approach position of the place frame. Similarly, this is 150 mm above the place frame. It then descends to the place frame and opens the gripper. It proceeds to return to ascend back to the approach position, where it waits for the next command.

We wrote two functions to accomodate pick and place. StateMachine.pick_up_block() adds waypoints that correspond to the first 4 steps and takes the pick frame as input. StateMachine.place_block() adds waypoints that correspond to the last 4 steps and takes the place frame as input.

IV. RESULTS

The following sections show the results of our project in verifying the algorithms for camera calibration, block detection, forward kinematics, robot control waypoint following, and the final competition.

A. Calibration

The following are the calibration matrices generated by various calibration methods.

Intrinsic Matrix (Factory Calibration):

$$\begin{bmatrix} 9.005 \times 10^2 & 0 & 6.614 \times 10^2 \\ 0 & 9.436 \times 10^2 & 3.611 \times 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

Intrinsic Matrix (ROS Calibration):

$$\begin{bmatrix} 9.289 \times 10^2 & 0 & 6.560 \times 10^2 \\ 0 & 9.009 \times 10^2 & 3.534 \times 10^2 \\ 0 & 0 & 1 \end{bmatrix}$$

Extrinsic Matrix (Apriltag Calibration):

$$\begin{bmatrix} 9.998\times10^{-1} & -1.755\times10^{-2} & -5.243\times10^{-3} & 8.781 \\ -1.697\times10^{-2} & -9.953\times10^{-1} & 9.556\times10^{-2} & 1.504\times10^{2} \\ -6.896\times10^{-3} & -9.545\times10^{-2} & -9.954\times10^{-1} & 1.001\times10^{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Extrinsic Matrix (Hand Measurements):

$$\begin{bmatrix} 9.99\times10^{-1} & -3.41\times10^{-2} & -7.25\times10^{-3} & 4.30 \\ -3.48\times10^{-2} & -9.77\times10^{-1} & 2.07\times10^{-1} & 1.23\times10^{2} \\ -5.05\times10^{-5} & -2.07\times10^{-1} & -9.78\times10^{-1} & 1.03\times10^{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homography Matrix:

$$\begin{bmatrix} 1.357 & -8.941 \times 10^2 & -2.484 \times 10^2 \\ 2.09 \times 10^{-2} & 1.140 & -7.006 \times 10^1 \\ 3.315 \times 10^{-6} & -1.046 \times 10^{-4} & 1 \end{bmatrix}$$

To validate the intrinsic and extrinsic matrices, we fed the grid points through equation (4) and projected them on the image. The projection is in Fig. 7.

B. Block Detection

To determine the accuracy of the visual block detection algorithm, blocks in all six colors (ROYGBV) were subsequently placed one at a time in locations along a grid in the workspace. Figure 8 shows a heatmap of these locations on the board, with the number of successfully detected blocks in each location. The number of successes ranged from 4-6 out of 6 blocks, with orange and blue being the only blocks that failed detection. The block detection showed an average success rate of 5.59 blocks per grid square and an overall success rate of 93.1% of blocks detected.

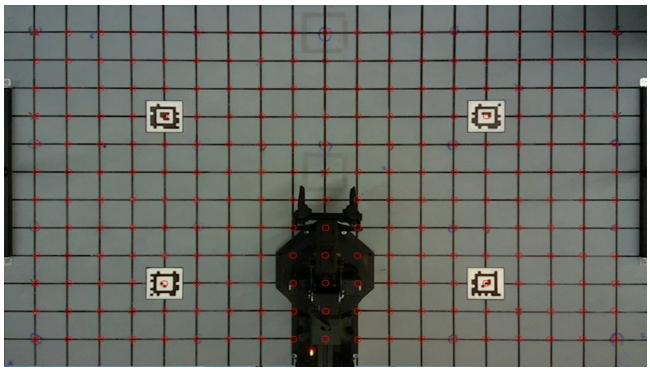

Fig. 7: Grid point projection with homography transform applied

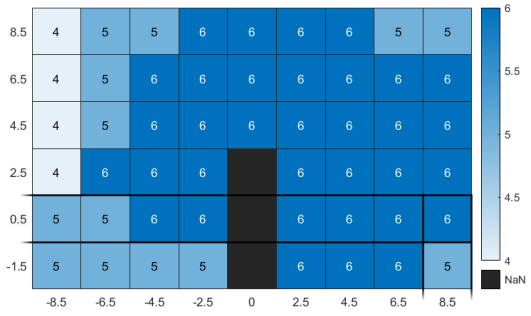

Fig. 8: Heatmap of Color Detection. Units are in 50mm.

C. Forward Kinematics Verification

In order to verify the accuracy of the forward kinematics derivation, the arm was manually moved to 11 points in a diagonal line across the workspace and the end effector was calculated from the joint positions. Table II shows the actual and calculated end effector poses, as well as the normed error between them. As shown, the forward kinematic error differed on the scale of 7-17 mm. The forward kinematic derivation was not used later in the competition, so this margin was deemed acceptable.

x_{des}	y_{des}	x_{act}	y_{act}	$\|\|e\|\|$
-400	-75	-393.6	-70.9	7.6
-350	-25	-335.1	-27.3	15.1
-300	25	-287.2	18.6	14.3
-250	75	-237.8	69.4	13.4
-200	125	-186.6	115.6	16.4
-150	175	-139.1	162.5	16.6
-100	225	-93	210.6	16.0
-50	275	-45.9	261.8	13.8
0	325	0.5	310.6	14.4
50	375	52.2	360.1	15.1
100	425	104.3	413.9	11.9

TABLE II: Forward Kinematic Verification Results (mm)

D. Teach and Repeat Block Cycling

We taught the arm to switch the position of two blocks by moving through an intermediate position and found that the procedure was repeatable to at least 10 times. Fig. 9 shows the position, obtained through forward kinematics, of the robot arm through one cycle. Fig. 10 shows the joint angle of each motor through the same cycle.

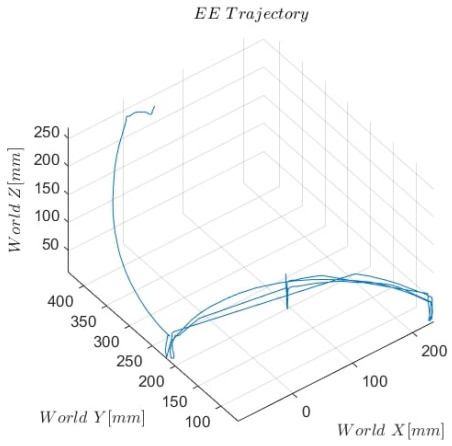

Fig. 9: Block cycling end effector trajectory

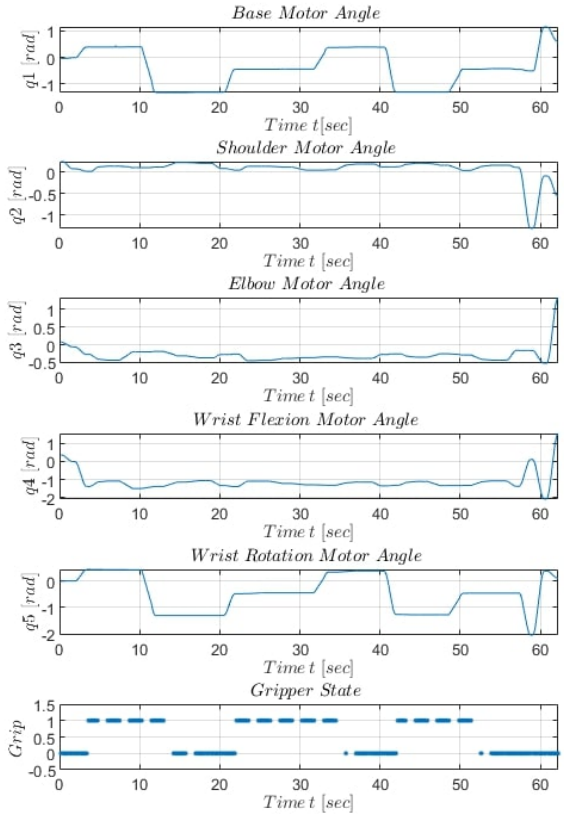

Fig. 10: Block cycling joint positions. A grip value of 1 represents a closed gripper. 0 is open gripper.

E. Final Competition

The final competition evaluated all aspects of the robot software stack that had been developed, excluding forward

kinematics since the robot position controllers were at the joint level, not the task level.

Event 1 consisted of detecting blocks, grasping them, and placing them on a side of the robot specified by their size. In order to fit more blocks next to the robot, we had each block be inserted with the grasper in a vertical orientation. This, along with the other trials, was our main verification metric for the inverse kinematics algorithm. The robot was able to successfully achieve level three on this challenge by picking up 8 out of 9 blocks and sorting them by size.

Event 2 was similar to Event 1, except it involved stacking the blocks in sets of threes rather than simply placing them beside the robot. Successfully completing this event required a tighter tolerance on the inverse kinematics solver, since the lateral position of the end effector when stacking blocks at various levels needed to remain consistent, otherwise blocks would fall. The robot was able to sucessfully complete level 2 by stacking 6 blocks of two sizes in stacks of three. The robot lacked the robustness to stack more of the smaller blocks at the next level.

Event 3 consisted of detecting both the color and size of blocks and placing them in a line of chromatic order. Our robot was able to mostly complete the second level, by lining up 6 smaller blocks and 4 large blocks in chromatic order. One large block was nearby the line, and the other was in the top left region, shown by Figure 8 to be less robustly detected. The robot was unable to complete the third level with distractor objects because, although it was successfully able to label and avoid most distractors, the tall cylinder was detected to be a small cube and was picked up. Our vision processing system consistently struggled to identify the correct outline orientation of the smaller blocks since, once filtered to a lower resolution, their contours resembled circles. However, the robot was still generally able to grasp them because smaller deviations in yaw of the gripper resulted in simply turning the block being grasped.

Event 4 consisted of stacking six small blocks in chromatic order. Due to time constraints in the overall competition, only the first level was attempted. Unfortuantely, the inverse kinematics algorithm was not consistent enough to stack the small blocks every time and, while all six were stacked in practice rounds, only a maximum of three were stacked in competition. The Event 4 algorithm first sorted the blocks on the left side of the robot base, then stacked them in color order. This two step method was easier to implement because the grasping script grabbed blocks in ascending order of distance from the robot base to avoid collisions while running several trajectories without feedback.

Finally, the bonus event was attempted but unsuccessful before the competition, due to a lack of force feedback and lower level control of the servo motors in carrying a larger tower.

Overall, this team placed second in the PM section competition with 848 points and exhibited successful behaviors in each aspect of the algorithmic stack.

V. DISCUSSION

A. Intrinsic Matrix Selection

The factory and ROS calibrated intrinsic matrices appear to be very similar, only differing by a little bit in any of the entries. We found that both intrinsic matrices performed similarly, but we obtained better grid projection with the factory intrinsic matrix. We found this surprising as the ROS calibrated one should be the most accurate. The most likely reason for the discrepancy between the two matrices is a slight change in lens position on the camera. This would mean that the ROS calibrated one would account for our current lens position while the factory calibrated one would have accounted only for what the position was in the factory, prior to any changes due to vibration or manual handling.

B. Calibration Performance

While the agreement in Fig. 7 suggests that our intrinsic and extrinsic matrices were fairly accurate, we experienced issues with the z values. Fig. 2 shows a colorized depth frame of the world z values. There is a clear trend in z values, where the top has slightly higher ones and the bottom of the grid has lower ones. We believe that this is due to noise in the depth sensor. To account for this, we zero the depth frame at the board. This results in Fig. 3. While we still had some issues further away from the board, we had an error of just 10mm in z for up to a stack of 4 big blocks, which corresponds to an expected height of 140mm. We found this acceptable for completing the events in the competition. However, this would pose problems for stacking beyond 4 blocks, such as for the bonus event. Further investigation into why the raw LiDAR readings are so poor would be necessary to make a more robust solution.

We had issues with orientation detection. Detected angles would vary by as much as $10°$, despite no movement. We believe that this was due to the segmented depth image coming out with rounded edges. Our implementation utilized cv2.minAreaRect(), but with a varying round edge, one can imagine how the rectangle that gets fitted could be moved around a bit. Morphological transformations designed more for edge detection would allow us to sharpen these edges and hopefully result in more stable orientation detection. Functionally the angle instability presented little issue, as most blocks would self align to the gripper, but some work on this could help us perform more fine movements. For instance, if the grid were densely packed, we would need to precisely grasp the desired blocks in the appropriate orientation to avoid disturbing other blocks.

C. Color Detection

The computer vision consistently detected red, yellow, green, and violet accurately. For most spots, orange and blue were also detected correctly; however, as can be seen in Figure 8 they fail in a considerable amount of spots. Orange and blue were misidentified as red and violet, respectively, with blue having the highest failure rate. We believe this was a result of inconsistent lighting conditions. They tended to fail in the areas with shade. This was likely a result of poor

tuning for nominal colors. LAB should abstract away the lighting issue by placing it in the L value. There was likely still some effect that shifted the A and B values, but we suspect that with better chosen nominal LAB color values, we could encapsulate the range of possible A and B values for each color better.

D. Block Type Detection

We found that each block could consistently be identified accurately. However, triangular blocks, depending on orientation, could be misidentified as small blocks or archs instead of as a distractor. We also never accounted for the possibility of cylinders being used as a distractor. The enclosed square of the cylinders used for competition were comparable in size to the small block face. This meant that we could never use cylinders as distractor objects, as they would consistently be identified as small blocks.

E. Inverse Kinematics

Our IK function occasionaly throws a position unreachable error when the end effector is in the down confriguation for blocks that are well within the dexterous workspace. To remedy this, the algorithm attempts to increase the wrist angle until the block position is reachable. However, this error still persists because of an unknown bug in the inverse kinematics function that could have been determined through further debugging. Alternatively, a numerical IK solver could have been implemented.

VI. CONCLUSIONS

Overall, our robot performed well, meeting most metrics for success. The vision system was able to detect most blocks, their orientation, their position, and their color, and the inverse kinematics solver and control system was able to grasp them and stack them with decent accuracy. Further tuning of the vision system with learning methods could have detected blocks at the edge of the workspace. Further debugging of the inverse kinematics solver could have improved the repeatability of grasping and stacking, but the control system would be limited to lighter loads until force feedback and feedforward torques can be returned from and applied to the joints.

REFERENCES

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

[2] A. Eggert and R. Fisher, "A comparison of four algorithms for estimating 3-d rigid transformations," 05 1998.

[3] J. Denavit and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, vol. 22, pp. 215–221, 06 1955.