Team 10 ROB 550 ArmLab Report

Rahul Agrawal, Oliver Bierman, Daksh Narang

***Abstract*—This report outlines the algorithmic design of a 5 degree-of-freedom robotic arm for manipulation. The objective is to detect blocks, stacked or unstacked in a workspace using a RGB-D camera, and apply path planning and kinematics to a robotic arm to pick and place the blocks. 3D calibration was successfully performed for transformation between image and world coordinates. Block detection was implemented and blocks of various colors and two sizes were detected along with their centroids and orientation. Inverse Kinematics and motion planning was implemented to grab and place blocks according to the task.**

I. Introduction

A 5-DOF RXarm is used for manipulation task of picking and placing/stacking blocks in the workspace. The project covers all the three aspects (sensing, acting, reasoning) of robotics. For acting, forward and inverse kinematics is implemented to generate the joint configuration given a goal position. For sensing, 3D image and workspace calibration is performed and blocks in the workspace are detected using OpenCV and RGB-D camera. For reasoning, state machines are implemented depending on the task required from the robot and path planning algorithms for each task are devised for smooth movements of the arm between the positions.

This report covers the methods to achieve such functionalities in section II, the performance in section III, summarizes the issues and future improvements in section IV, and concludes in section V.

II. Methodology

A. Basic Motion and Camera Calibration

1) PID Gains: The P, I, D values for various joints were kept unchanged. The belief in the accuracy of gains was bolstered using Forward Kinematics. Certain joint angle configurations were fed into the system and workspace coordinates were recorded as reported on the GUI. These were in turn compared with the actual known workspace coordinates. The comparison did not yield a major difference between the two values. Hence, default PID gains were used.

2) Camera Intrinsic Calibration: To convert from the camera to image frame we need to determine the intrinsic camera matrix. We used a checkerboard along with ROS camera calibration package running on realsense node to get the intrinsic matrix for Realsense Camera following the tutorial [1]. On running the calibration node, the checkerboard was moved in the camera frame to capture data. The checkerboard was moved through multiple orientations, ensuring we cover the entire field of view of the camera. At each position, the board was held still for some time until the image was highlighted in the calibration window. The board was also tilted in some orientations to account for skew.

The Results section mentions our intrinsic matrix alongside the factory intrinsic matrix.

3) Camera Extrinsic Calibration: Extrinsic matrix facilitates converting the world to camera frame. The first approach used to calculate the extrinsic matrix involved measuring the camera location with respect to the base frame along the various axis using a measuring tape.

The second and a much better approach was using Apriltags, which is discussed in the following section. Extrinsic matrix using both methods is reported in the Results section.

B. Automatic/Semi Auto Camera Workspace Calibration

1) Workspace Calibration: A workspace calibration was performed to convert coordinates from image frame to world frame. To perform the calibration, an auto-calibration routine was implemented using Apriltags visible in the workspace. Apriltags are fiducial markers whose coordinates in the camera frame can be recorded using the ROS node apriltag_ros. The coordinates of the apriltags in the world frame are also recorded. The coordinates of apriltags in the camera frame were converted to image frame using

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = (1/Z_c) * K * \begin{bmatrix} I|0 \end{bmatrix} * \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{1}$$

where u, v are the pixel values in image frame, K is the intrinsic matrix, I is identity matrix, and X_c, Y_c, and Z_c are the coordinates of apriltags in camera frame.

The recorded coordinates for 4 apriltags in the workspace were used along with the distortion coefficients of the intrinsic matrix as inputs to SolvePnP algorithm of OpenCV. The iterative algorithm of SolvePnP was used. The algorithm returns a rotation and a translation vector (T). This rotation vector is used with Rodrigues' rotation formula to compute the rotation matrix (R). The extrinsic matrix H is

$$H = \begin{bmatrix} & R & & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

In the GUI, the calibration can be performed simply by clicking the PnP calibration button. After the calibration is completed, a message window pops up stating its success. The display under the video in the GUI then show the workspace coordinates corresponding to the mouse location as we hover over the video. To convert the pixel coordinates to world coordinates the following equations are used

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = Z_c(u,v) * K^{-1} * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{3}$$

$$\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} = H^{-1} * \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} \tag{4}$$

where X_w, Y_w, and Z_w are the coordinates in world frame.

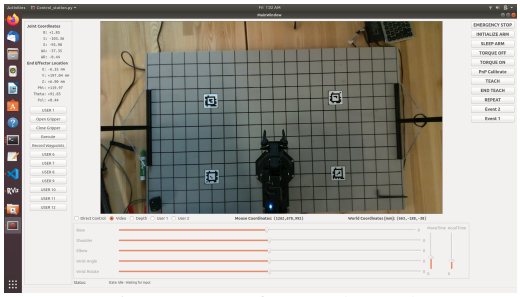

Fig. 1: GUI of control panel

2) Teach and Repeat: Teach and repeat is a way of programming robotic arms to do repetitive task. The torque of all the motors is set to zero and the user physically moves the arm through the workspace. The GUI is shown in Fig. 1. To start the process, the user presses the "Teach" button to enter the teach state. A "record waypoint" button was implemented in the GUI which records the desired waypoints by the user. After the teaching is completed, the user presses the "Teach End" button to end teaching process and is asked to turn the torque back on. On clicking the "Repeat" button, the system enters the repeat state. In this implementation, while teaching, the user is asked to have two intermediate waypoints between every current and goal waypoint. With this, the gripper is automatically alternated between opening and closing at every third waypoint and also ensures smooth transition of the arm between waypoints. The system enters the idle state again after the repeat process is completed.

C. Block Detection and Kinematics

1) Block Detection: To detect blocks, first the region outside the board and the arm is masked out from the image feed. The depth value is used to set upper threshold to 957 and lower threshold to 807, and these thresholds are used to detect contours in the image. To filter out the noise a morphological opening filter, which is erosion followed by dilation is applied with a kernel size of 13*13 (all ones) before detecting the contours. It is useful for reducing the noise outside the contours of interest. For each contour detected, the mean of the RGB values of the countour is checked for the nearest color amongst red, orange, yellow, green, blue, purple, and pink to determine the color of the block. The orientation of the block is computed using minAreaRect() function of OpenCV. Moments() function is used to get the size (small or large) of the block and to determine the centroid of the block using

$$C_x = M_{10}/M_{00} \tag{5}$$

$$C_y = M_{01}/M_{00} \tag{6}$$

where C_x and C_y are the centroids of the block in image frame and M is the moment.

Finally, the centroids of the blocks detected in image frame are converted to world frame using the equations described in II.B.1.

2) Forward Kinematics: The Denavit-Hartenberg (DH) convention was used for establishing the location and orientation of the frames on the robotic arm. The schematic of the arm and DH parameters are shown in Fig. 2 and Table I.

Once the DH parameters were known, the location and orientation of the grippers origin was found

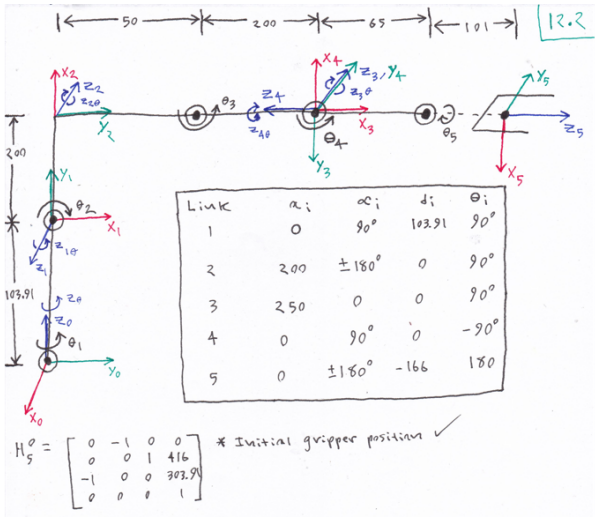

Fig. 2: Schematic of arm

DH Table				
Link	a_i	α_i	d_i	θ_i
1	0 mm	90°	103.91 mm	90°
2	200 mm	180°	0 mm	90°
3	250 mm	0°	0 mm	90°
4	0 mm	90°	0 mm	-90°
5	0 mm	180°	-166 mm	180°

TABLE I: DH Table

by calculating the homogeneous transformation matrix H_5^0. The formulas for this calculation can be seen below:

$$A_i = \begin{bmatrix} c_{\theta i} & -s_{\theta i}c_{\alpha i} & s_{\theta i}s_{\alpha i} & \alpha_i c_{\theta i} \\ s_{\theta i} & c_{\theta i}c_{\alpha i} & -c_{\theta i}s_{\alpha i} & \alpha_i s_{\theta i} \\ 0 & s_{\alpha i} & c_{\alpha i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$H_5^0 = A_1^0 A_2^1 A_3^2 A_4^3 A_5^4 \tag{8}$$

3) Inverse Kinematics: Our Inverse Kinematics solution diverges from what was taught in class. The method to be explained was developed with the assumption that the arm is always in the elbow up position and that we are never trying to grab blocks out of the air. The order in which we calculate the joint angles is θ_5, θ_4, θ_1, θ_2, and finally θ_3.

We begin by first calculating θ_5. To find θ_5 we first need to know the orientation of the block θ_c. This is found in the block detection code with OpenCV. We then need to find the angular difference between the blocks orientation and the arms orientation in the XY plane. That will be θ_5. There are four possible equations that can be used depending on what quadrant the block is in. Those equations are listed below. For reference, in kinematics.py θ_5 is calculated on lines 331-347.

$$Q1 : \theta_5 = tan^-1(x, y) + \theta_c * (\pi/180) \tag{9}$$

$$Q2 : \theta_5 = -tan^-1(x, y) + \theta_c * (\pi/180) \tag{10}$$

$$Q3 : \theta_5 = tan^-1(x, y) + \theta_c * (\pi/180) \tag{11}$$

$$Q4 : \theta_5 = -tan^-1(x, y) + \theta_c * (\pi/180) \tag{12}$$

The second joint angle to be found is θ_4. To find θ_4 we used linear interpolation. This was done by defining a linear interpolation table between the minimum and maximum reach distances of the arm. We assumed the minimum distance was 125mm and the maximum distance was 375 mm. Between 125mm-375mm we found real values for θ_4 that would go into the interpolation table. The real value angles found were with the grippers fingers in contact and flat against the board. If the arm needed to reach a block passed 375mm we commanded θ_4 = 0° (straight wrist). The calculation begins by finding the shortest line distance r between the centroid of the block and the base frame origin. We then determine where r falls within the interpolation table and interpolate what θ_4 should be at that distance. The interpolation table is table II and formulas used are (13) and (14). For reference, in kinematics.py θ_4 is calculated on lines 236-257.

θ_4 Interpolation Table	
r	θ_4
125 mm	-51.94°
175 mm	-52.91°
225 mm	-56.60°
275 mm	-61.70°
325 mm	-70.66°
375 mm	-84.55°
425 mm	-0°

TABLE II: θ_4 Interpolation Table

$$r = \sqrt{(x-0)^2 + (y-0)^2} \tag{13}$$

$$\theta_4 = \theta_{4,1} + \frac{\theta_{4,2} - \theta_{4,1}}{r_2 - r_1}(r - r_1) \tag{14}$$

The third joint angle calculated is θ_1. There are four possible equations that can used depending on the quadrant the block is in. There are also four special cases. $\theta_1 = 0°$ if its between the 1st and 4th quadrant. $\theta_1 = 90°$ if its between the 1st and 2nd quadrant. $\theta_1 = 180°$ if its between the 2nd and 3rd quadrant. $\theta_1 = 270°$ if its between the 3rd and 4th quadrant. The equations to calculate θ_1 can be seen below. For reference, in kinematics.py θ_1 is calculated on lines 215-234.

$$Q1 : \theta_1 = tan^-1(y/x) \tag{15}$$

$$Q2 : \theta_1 = tan^-1(y/x) + 180 \tag{16}$$

$$Q3 : \theta_1 = tan^-1(y/x) + 180 \tag{17}$$

$$Q4 : \theta_1 = tan^-1(y/x) \tag{18}$$

Finally, the fourth and fifth joints calculated are θ_2 and θ_3. For this calculation we first need to know θ_5,θ_4, and θ_1. The second thing we need to know is the range of angles to sweep through for the shoulder and elbow joint. These values were found by moving the arm to its minumum and maximum reach distance in the elbow up configuration. For the shoulder, the range used was ($-34.19°$,$64.25°$). And for the elbow, the range used was ($-61.96°$,$63.46°$). Now that we know (θ_5,θ_4,θ_1), the ranges to sweep through, and the x, y, and z position of the block, we can use Forward Kinematics to find the joint angles. We used two For-loops and swept through 51,408 angles at steps of $0.5°$. For each angle we calculated H_5^0 and compared d_5^0 to the x,y, and z position of the block. If the x,y,and z position of the grippers origin was within 1.2 mm of the x,y, and z position of the centroid of the block, then we take those two values as θ_2 and θ_3. If the position could not be calculated we did an even finer search by sweeping through 115,668 angles at steps of $0.33°$. If the position can still not be reached then we print to screen the arm cannot reach that position. For reference, in kinematics.py we calculate θ_2 and θ_3 in lines 369-429.

4) Motion Planning: For Motion planning of RX200 arm, click to grab and click to place was implemented. The coordinates of the mouse pointer are recorded in the image frame on clicking on a block. These coordinates are transformed to world frame. The coordinates in the world frame are input into the Inverse Kinematics to get the joint angles. Similar to teach and repeat, there are two intermediate waypoints between every current position and target for smooth transition. The arm is first moved to the initialize position and then to a position vertically above the block. The gripper alternates between closing and opening with every click. The first click closes the gripper allowing the arm to grab the block on reaching the pick location and the second click opens the gripper after the arm reaches the destination.

D. Autonomy

This section explains the planner to control the motion of the arm and implement states to control the arm for picking and placing blocks. Initially, the gripper is in open state. The block detection algorithm is used as described in II.C.1 to get the centroids of the all the blocks in the workspace in world frame. The destinations of small and large blocks are set according to the task in the competition. For each block detected, the joint positions are determined using Inverse Kinematics with the centroid detected as the end effector position. First, the arm is initialized and then set to a position 40 mm above the goal position (the centroid of the block). Then the waypoints are set to the goal position and the gripper is closed to grab the block. Similarly, the waypoints are set to 40 mm above the block again and finally to the destination via an intermediate waypoint. These intermediate waypoints ensure smooth transition between the targets. The gripper is opened to place the block and this process is repeated for every block detected. Our team implemented states for Event 1 (level 2) and Event 2 (level 2) of the competition. Both the states have the same algorithm as described above and only the destinations are changed.

III. Results

A. Basic Motion and Camera Calibration

1) PID Gains: The PID gains are mentioned in Table III:

Joint Gains	P	I	D
Waist Gains	640	0	3600
Shoulder Gains	800	0	0
Elbow Gains	800	0	0
Wrist Angle Gains	800	0	0
Wrist Rotate Gains	640	0	3600
Gripper Gains	640	0	3600

TABLE III: Joint PID Gain Values

The PID gains were not changed, because they were almost accurate with the motor angles being reported within a $\pm 1°$ of the desired value.

2) Camera Intrinsic Calibration: We performed 5 calibration rounds, and the average calibration intrinsic matrix and distortion matrix is as follows:

$$K = \begin{bmatrix} 912.136 & 0 & 653.24 \\ 0 & 913.056 & 348.001 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

$$D = \begin{bmatrix} 0.12045 & -0.21756 & 0.00183 & -0.0071 \end{bmatrix} \tag{20}$$

The factory calibration intrinsic matrix and distortion matrix is as follows:

$$K_f = \begin{bmatrix} 896.1685 & 0 & 657.4849 \\ 0 & 896.066 & 350.7544 \\ 0 & 0 & 1 \end{bmatrix} \tag{21}$$

$$D_f = \begin{bmatrix} 0.17 & -0.51 & -0.0022 & 0.00015 & 0.47 \end{bmatrix} \tag{22}$$

There is a slight difference between the two intrinsic matrices. This can be attributed to the quality of images of the checkerboard used for calibration and the process being manual. It also depends on the lighting conditions being used, the size and position of the checkerboard pattern, lens aperture, and camera and lens quality. In addition, there might be geometric distortion due to the lens and a skew such that the center of the image plane and the principal point do not coincide.

We used our intrinsic matrix, it depicted a lower variation in coordinates across the workspace with better depth uniformity.

3) Camera Extrinsic Calibration: The Nominal Extrinsic Matrix calculated by measuring the mounting geometry of the camera is as follows:

$$H_{Nominal} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 175 \\ 0 & 0 & -1 & 967.58 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (23)$$

We adopted the auto calibration methodology using April tags for a better estimate of extrinsic matrix which turned out to be as:

$$H = \begin{bmatrix} 0.9995 & -0.0058 & -0.028 & 4.403 \\ -0.0052 & -0.999 & 0.0215 & 179.009 \\ -0.0284 & -0.0213 & -0.9993 & 968.801 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (24)$$

The matrices are not the same. We cannot rely solely on the nominal extrinsic matrix values because they are calculated by manually measuring distances which can be a source of error. Other reasons might be the quality of Lidar and camera, which affect camera resolution and depth perception, that is it gives less accurate readings as the object moves farther away from the camera. Moreover, we have limitations posed by the PnP solver and the pattern of Apriltags. Also, these fiducials should be fully exposed during the calibration procedure, otherwise it does not report accurate values.

B. Automatic/Semi Auto Camera Workspace Calibration

1) Workspace Calibration: The intrinsic matrix K is

$$\mathrm{K} = \begin{bmatrix} 912.136 & 0 & 653.24 \\ 0 & 913.056 & 348.001 \\ 0 & 0 & 1 \end{bmatrix}$$

To verify that the calibration is correct, the camera was rotated and the auto-calibration was performed by pressing the calibration button in the GUI. With the camera rotated, we recorded the reported position (x,y,z) of the center of the top of a stack of large blocks placed at the following locations for a stack size of [0,1,2,4,6] blocks:

(0,175), (-300, -75), (300, -75), and (300, 325)

All the values are in mm.

No. of blocks	(0, 175)	(-300,-75)	(300, -75)
0	(2,176,-7)	(-307,-83,-8)	(309,-83,-14)
1	(3,176,31)	(-310,-86,26)	(311,-84,21)
2	(1,177,69)	(-308,-86,64)	(314,-82,57)
4	(4,177,146)	(-307,-84,142)	(317,-82,135)
6	(7,177,220)	(-303,-81,217)	(318,-81,210)

No. of blocks	(300, 325)
0	(306,326,-8)
1	(308,328,29)
2	(311,330,67)
4	(310,330,143)
6	(311,331,220)

TABLE IV: Workspace calibration

The values reported in Table IV confirms the accuracy of the calibration. Using the distortion coefficients from the intrinsic calibration in the SolvePnP function helped improve the accuracy of calibration.

2) Teach and Repeat: The arm was able to repeat the cycle six times. With each iteration, the block is not placed precisely at the expected location and the orientation changes when the block is dropped. This error keeps accumulating as the cycles progress and eventually the arm is unable to grab the block.

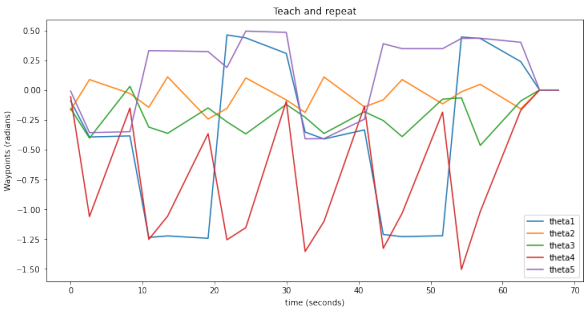

Fig. 3: Plot of waypoints vs. time

Fig. 3 shows the plot of waypoints vs. time for 1 cycle of repeat.

C. Block Detection and Kinematics

1) Block Detection: To verify the accuracy of the block detector, the detected centroid position was

Detected Centroid (mm)	World coordinate (mm)
(-298, 324, 35)	(-300, 325, 37)
(-309, -75, 37)	(-300, -75, 37)
(202, -77, 28)	(200, -75, 37)
(199, 320, 28)	(200, 325, 37)
(-51, 372, 32)	(-50, 375, 37)
(48, 422, 28)	(50, 425, 37)
(-408, -80, 37)	(-400, -75, 37)
(-202, 225, 33)	(-200, 225, 37)

TABLE V: Verification of block detection

recorded against some known positions on the board. The recording is tabulated in Table V.

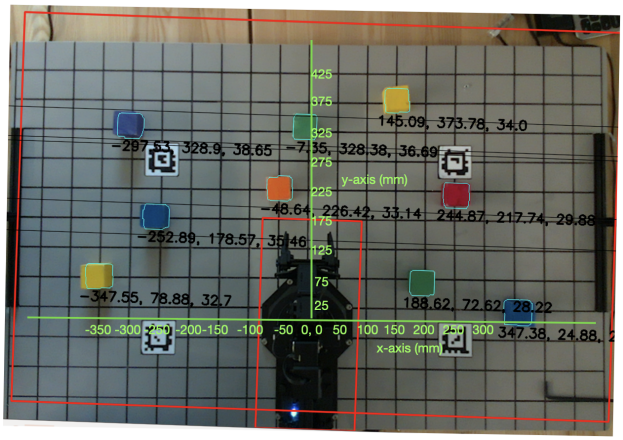

Fig. 4: Plot of centroid detections

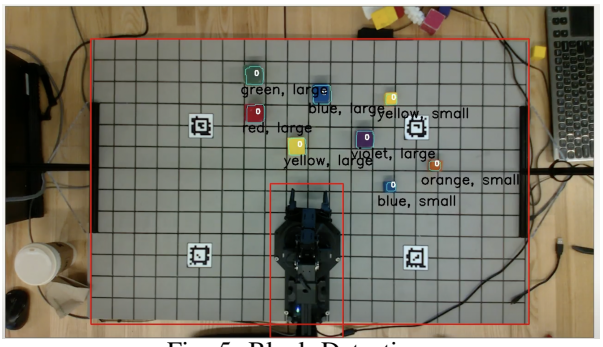

Fig. 5: Block Detections

Fig. 4 depicts a plot of the workspace where blocks are placed at different locations and their detected centroids (x,y, and z) are shown. Fig. 5 shows provides evidence of the block detection with their color and sizes labeled.

2) Forward Kinematics: To verify that the FK equations were producing the desired workspace coordinates we used three different methods. First, we verified whether or not H_5^0 was being calculated correctly. This was done by calculating H_5^0 for the arms initialized position and comparing it to the base frame in the schematic of the arm. By looking at R_5^0 and d_5^0 in the homogeneous transformation matrix, and using the arms known dimensions and the base frames orientation. We were able to confirm the grippers origin was in the correct position and orientation. We then checked the grippers actually position when the arm was experiencing sag from gravity. Our X and Y coordinates were both within 3mm of the ideal values. Our Z value was off by about 20-25mm due to gravity causing the arm to sag.

The second method of verification was to calculate the FK of the arm for all ten way-points in section 1.2 of the arm lab document and then plot them in a 3D plotter. We ended up calculating every frame for all ten way-points. In total, fifty frames were plotted in the 3D plotter. We then compared the plotted frames with a video we had of the arm moving through all ten way-points. This method allowed us to verify the arm was in the correct quadrants, and it also allowed us to see the configuration of the arm at each way-point.

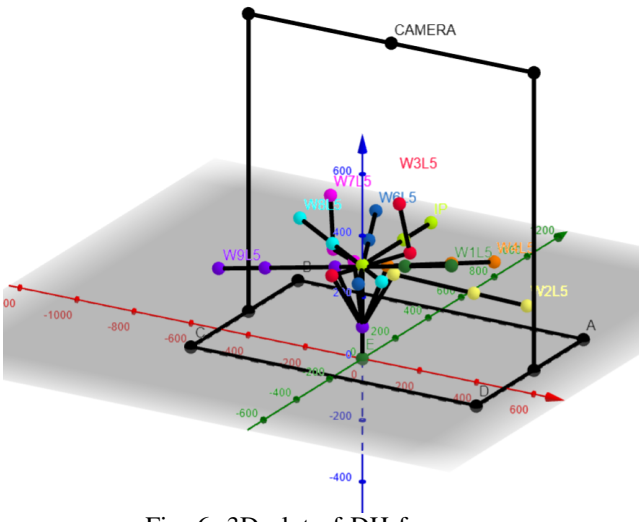

Fig. 6: 3D plot of DH frames

The last method of verification was done by moving the arm to known coordinates on the board. This was done by turning the torque off for all the joints, then moving the gripper to the known locations, and placing the tip of the gripper in contact with the board. We then calculated the error between the known coordinates and our calculated coordinate's. Our X and Y values were anywhere from 3-11mm off. Our Z values had quite a bit of variation in them. The values were negative when really close to the base. And slowly became more positive as the arm extended farther and farther out, eventually turning positive. We compensated for this later on by using linear interpolation on the Z values.

FK Verification		
Pose	x,y,z (Actual in mm)	x,y,z (Estimate in mm)
1	(300,-75,25)	(303.22, -64.68, 19.55)
2	(-300,-75,25)	(-296.56, -84.52, 15.6)
3	(0, 175, 62)	(-10.77, 167, 19.66)
4	(0, 125, 25)	(-4.5, 114.6, -31.56)

TABLE VI: FK Verification

Table VI shows actual position vs the estimated position.

3) Inverse Kinematics: Our IK solution for the arm worked and we were able to pick and place blocks with click to grab. We were also able to complete Event One level 2 and Event Two level 2 in the competition. It took our arm about 2 minutes and 16 seconds to complete each event. We ended up getting 350 points overall for those two events. Some drawbacks were that the arm was slow to calculate θ_2 and θ_3. It would take about 1.4 seconds to finish this calculation. However, for the events we competed in, it was not necessary for the arm to calculate the joint positions quicker. For faster calculation time we were planning on using the bisection algorithm. That most likely would of made our time to calculate IK much faster. Even though our IK solution was odd compared to the standard solution, the arm was able to complete its tasks in a timely manner with relatively good precision.

D. Autonomy

In the competition, the robotic arm successfully completed level 2 of Event 1. 6 blocks, 3 large and 3 small of random colors were detected by the block detector and picked and placed by the robotic arm at the required destinations. The team earned 200 points for Event 1. For future work, the case where blocks are stacked need to be taken into account. One possible method is to use the depth value of the detected centroid of the top of the stack to determine whether it is a stack of large or small blocks. Furthermore, the block detection algorithm needs to run continuously to update the detections in the system.

For Event 2, the robotic arm attempted level 2 and completed it with 5 blocks. The team earned 150 points for this event. A small block was not stacked in the right place because of inaccuracies during placing the block on a stack from the top. One possible improvement is to grab blocks from the side rather than from the top.

We tried participating in Event 3, 4, and bonus event but the arm still requires work as the logic for detecting stacked blocks needs further tuning and refinement.

IV. Discussion

The objective was achieved as the system automatically detects blocks in the workspace and the robotic arm picks and places the blocks. The camera was calibrated for intrinsic matrix using checkerboard and extrinsic using PnP solver and April tags. As checkerboard calibration is a manual process, there is scope of improvement and more data might lead to more accuracy. Similarly, using more april tags instead of 4 can increase accuracy of extrinsic calibration. In the current implementation of teach and repeat cycle, the user is restricted to two intermediate waypoints but this can be made a variable with the user made to input the waypoint at which the gripper must be opened or closed. The block detector has high variance of error in the z coordinate which can be improved with adding offsets. As mentioned in the explanation of inverse kinematics, the method used did not follow the lectures and there is scope of improvement in computation time. Detection and planning for stack of blocks needs further tuning as discussed in III.D.

V. Conclusion

In summary, a system of robotic arm and RGB-D camera was developed to automatically detect blocks in a workspace and pick and place the blocks. The system uses concepts in all several aspects of robotics from kinematics, computer vision, path planning, to rigid body transformations, and state machines. The methodology of implementation were discussed and the results were presented in II and III. Possible future improvements were discussed in IV.

References

[1] ROS.org. Checkerboard calibration. [Online]. Available: http://wiki.ros.org/camera_calibration/Tutorials/MonocularCalibration