ROB 550: Armlab Report - Team 9

Salman Javed, Manas Jyoti Buragohain, Haochen Wu
{salmanja, manasjb, haochenw}@umich.edu

***Abstract*—This report documents the process of designing and implementing a 6 DOF robotic arm to perform various tasks such as stacking blocks, placing blocks in a line, etc. To enable the arm to pick and place blocks, a gripper design was proposed and 3D printed for attachment to the robot arm. Additionally, achieving this objective required the integration of kinematics, perception, motion planning and reasoning. The perception stack of this project was based on using Kinect depth and RGB camera for detection and classification of blocks based on color and stack height. The perception stack feeds block locations to the inverse kinematics module which calculates the joint angles and passes them to trajectory planning module for smoothing the path. The reasoning stack governs the actions the robot arm needs to execute to achieve the desired objective. *Keywords*: Robot arm, kinematics, perception, motion planning, reasoning**

I. Introduction

Robotic arms are an integrated system comprising of various components which build the robot structure, help ensure movement, execute various tasks and detect blocks. The project covers the basis of an robotic manipulator arm including mechanical design, perception, kinematics, motion planning, and trajectory smoothing. The robot structure comprises of 3D printed components which form its links. The joints are formed through use of 6 servo-motors. Three MX-28 servo-motors drive the base joint, shoulder joint and elbow joint. Two AX-12 servo-motors form the wrist joint. Two XL-320 servo-motors are used for the actuation of the gripper. These servo-motors communicate with each other through asynchronous serial communication. The gripper is made using 3D printed components. Block detection and classification task is achieved through the use of Kinect camera which provides RGB and depth images. The software for running the robot arm is created using Python in an Ubuntu environment. The software improves user friendliness through provision of GUI which can be used for camera calibration, selection of tasks to be performed, determination of end-effector location and joint angles. In this report, Section II describes the implemented methodology with reference to gripper design, trajectory smoothing, kinematics, perception and motion planning. Section III discusses the result of these implementation methods and the competition. Finally, section IV and V include our discussion and conclusions respectively.

II. Methodology

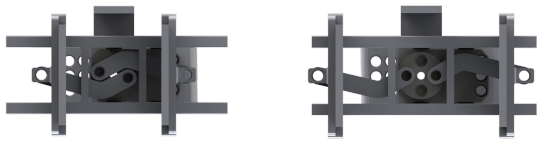
A. Gripper Design

The objective of our gripper design was to make an end-effector as light as possible while providing all the essential functionalities. Our gripper had the following characteristics:

- 3D printed (ABS material) gripper with two servo-motors to provide additional degrees of freedom
- Simple design with minimum material and optimized slots to reduce weights
- Additional plate attached to the gripper to adjust the block position

The gripper design was initiated by the selection of mechanism to toggle the gripper. We considered various ways to toggle the gripper such as: using a two-bar linkage, rotational linkage and cam actuated plate. We chose the rotational linkage with actuation in the horizontal plane as shown in Figure 1. This decision was based on the cons of other designs. For eg. the cam actuated mechanism needed additional gripping force for picking blocks, two-bar linkage complicated the design. Additionally, we did not actuate the arms in the vertical plane since this would expose the arms and might lead to interaction with objects during gripping.

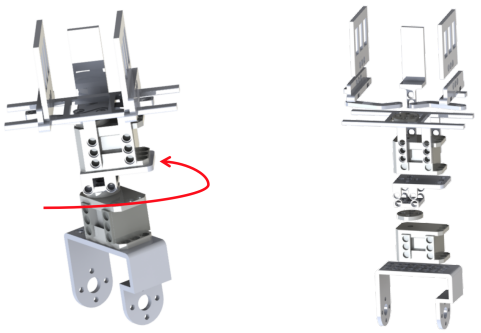
We also made choices regarding the degrees of freedom to be provided. Our initial gripper utilized only one motor to toggle the arms of the end effector. However, we later realized that this limited the orientations in which the blocks could be grabbed. This problem was overcome by the addition of another motor which enabled motion as shown by the red arrow in Figure 2.



(a) Closed position (b) Open position

Fig. 1: Rendered top view for the gripper open/close positions.

In terms of the design aspects, we kept the wrist center aligned with the vertical axis of the base frame for simplicity in kinematics. Our initial component connecting the AX and XL motor had limited holes for positioning



(a) (b)

Fig. 2: Rendered view for the gripper design. (a) The gripper with an additional DOF for orientation (b) An exploded view

the servo-motor at the desired location. We overcame this problem through extension of the piece and this enabled us to install the motor along the vertical axis. Additionally, we added a supporting piece which formed the third arm of the gripper to enable us to adjust the block position and orientation for more accurate block lining and stacking. To reduce the weight of the system, we removed material from the gripper arms and choose to create sliding rails with minimum length required for the movement of gripper arms. This was all done while considering that the printed components should not become fragile.

B. Trajectory Smoothing

Without trajectory smoothing, the arm would go from the current configuration to the goal configuration directly with the set speeds for all joints. Such trajectory would suddenly stop the arm at the goal configuration and potentially throw the block away due to its inertia. In order to resolve the problem, we desired a smooth trajectory which limited the accelerations of all joints to achieve less wear, reduced vibrations in motors while increasing overall precision.

To improve the smoothness of transformation between each configuration, a Quintic polynomial spline was used. This method limited the jerk by creating a continuous acceleration trajectory. For Quintic polynomials, the trajectories was defined as follows:

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \quad (1)$$

$$v(t) = \dot{q}(t) = a_1 + 2a_2 t + 3a_3 t^2 + 4a_4 t^3 + 5a_5 t^4 \quad (2)$$

$$a(t) = \ddot{q}(t) = 2a_2 + 6a_3 t + 12a_4 t^2 + 20a_5 t^3 \quad (3)$$

There are six unknowns, $a_0, \ldots, a_5$, in the Quintic polynomial trajectory equations and we have six known boundary conditions, $q_0, q_f, v_0, v_f, a_0, a_f$ at t_0, t_f, where q, v, a denote configuration, velocity, and acceleration respectively and the subscripts $0, f$ denote the initial and final conditions. The above equations can be put in matrix form as shown below:

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_0 & 12t_f^2 & 20t_f^3 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} q_0 \\ v_0 \\ a_0 \\ q_f \\ v_f \\ a_f \end{bmatrix} \quad (4)$$

Then the parameters a_i could be solved by multiplying the inverse of the square matrix on both sides.

$$x = A^{-1}B \quad (5)$$

For this project, with the given current configuration q_0 and the goal configuration q_f, the initial and final velocities and accelerations were set to be zero $v_0 = v_f = a_0 = a_f = 0$. The trajectory duration, $T = t_f - t_0,\ t_0 = 0$ was determined by first finding the max joint displacement $max(\Delta q)$ between two configurations and dividing the max joint displacement by the desired traverse speed v_d. $T = t_f - t_0 = max(\Delta q)/v_d$. With this quintic polynomial trajectory smoothing method, a smooth trajectory with limited jerks was produced.

C. Kinematics

1) Forward Kinematics: Forward Kinematics is the transformation from the configuration space $\Theta = \{q_1, q_2, q_3, q_4\}$ to the workspace $\{x, y, z, \phi\}$. That is, given the joint angles of robot, we can determine the position of the end effector of the robot. Techniques such as Denavit-Hartenburgh convention and Product of Exponentials exist for implementation of forward kinematics. We chose to proceed with DH convention since it is a fundamental technique in the analysis of kinematic chains.

The base of the robot was chosen as the global frame. The subsequent local frames were assigned as per DH convention rules and are shown in Fig. 3. It should be noted that the rotation of the gripper and its gripping motion does not affect the end effector position and orientation. Therefore, these joint angles were not considered in forward kinematics. The DH table and associated parameters: θ, d, a, α are shown in table I.

Each row in the DH table signifies a homogeneous transformation matrix (A_i) which is a product of four basic transformations as show in Eq. 6. The transformation from the end effector to the global frame can be obtained by premultiplying the transformation matrices as shown in Eq. 7:

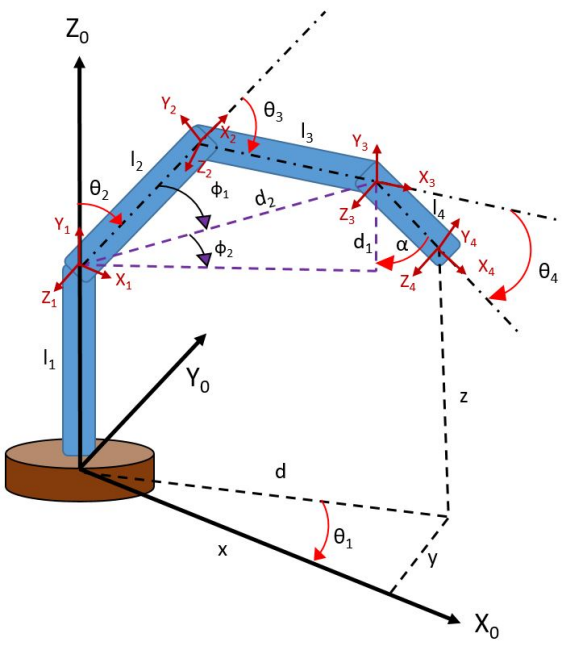


Fig. 3: Schematic for forward and inverse kinematics

TABLE I: Denavit-Hartenburgh Table

Joint No.	θ **(rad)**	**d (mm)**	**a (mm)**	α **(rad)**
1	θ_1	l_1	0	$\frac{\pi}{2}$
2	$\theta_2 + \frac{\pi}{2}$	0	l_2	0
3	θ_3	0	l_3	0
4	θ_4	0	l_4	0

$$A_i = Rot_{z,\theta_i} Trans_{z,d_i} Trans_{x,a_i} Rot_{x,\alpha_i} \quad (6)$$

$$T_0^5 = A_0^1 A_1^2 A_2^3 A_0^4 \quad (7)$$

2) Inverse Kinematics: Inverse Kinematics is the transformation from the workspace $\{x, y, z, \phi\}$ to the configuration space $\Theta = \{q_1, q_2, q_3, q_4\}$. That is, given the end effector location of the robot, we can determine the joint angles of robot. The inverse kinematics problem can be solved using geometry to derive the equations. However, the solution to these set of equations are not unique. Steps have been taken to limit the solution to a unique value (elbow-up configuration). Table IV in Appendix C provides the description for symbols used in inverse kinematics.

The inverse kinematic equations are solved within the `IK` function in `kinematics.py` file. Line numbers referring to this python file are mentioned below in {`LXXX`} format.

Firstly, it is checked if the desired location is within the workspace of the robot. This distance of the end effector from the base joint in the XY plane is given below {`L119`}:

$$d = \sqrt{x^2 + y^2} \quad (8)$$

This is followed by calculation of the rotation angle of the base joint θ_1 given by {`L124`}:

$$\theta_1 = atan2(y, x) \quad (9)$$

To solve for rotation angle of the elbow joint θ_3 we need to apply the Law of Cosines {`L131/L138`}:

$$\theta_3 = \pi - acos(\frac{d_2^2 - (l_2^2 + l_3^2)}{2 l_2 l_3}) \quad (10)$$

Solving the above equation requires parameter d_2 which can be obtained by applying Pythagoras theorem {`L130/L137`}:

$$d_2 = (d_1)^2 + (d - l_4 \sin\alpha)^2 \quad (11)$$

If z> l_1:

$$d_1 = l_4 \cos\alpha + (z - l_1) \quad (12)$$

If z< l_1:

$$d_1 = l_4 \cos\alpha - (l_1 - z) \quad (13)$$

The rotation angle of the shoulder joint θ_2 is given by {`L143`}:

$$\theta_2 = \pi/2 - (\phi_1 + \phi_2) \quad (14)$$

where ϕ_1 {`L142`} and ϕ_2 {`L129/L136`} are given by:

$$\phi_1 = \theta_3 - acos(\frac{l_2^2 - (d_2^2 + l_3^2)}{2 d_2 l_3}) \quad (15)$$

$$\phi_2 = atan2(d_1, d - l_4 \sin\alpha)) \quad (16)$$

The last step involves solving for rotation angle of the wrist joint θ_4 as follows {`L147`}:

$$\theta_4 = \frac{\pi}{2} - (\theta_2 - \phi_1 - \phi_2 + \alpha) \quad (17)$$

D. Perception

We utilized the Kinect sensor, which provides RGB and depth data, to locate the blocks in the world plane and precisely identify the color of the blocks. We detailed this process as a sequence of 3 steps: camera calibration, block detection and color identification.

1) Camera Calibration: The objective of this step was to find the correlation between any pixel location in the RGB image to a real world coordinate. We achieved this by finding a set of transformation matrices between the RGB and depth images, RGB image and camera frame and the final transformation between the camera frame and world frame.

The RGB and depth images from Kinect do not completely overlap due to the offset in their location

and difference in magnification between the two sensors. Hence, we began by registering the transformation between these images. This relation can be expressed as an affine transformation between the pixels, which is expressed as,

$$\begin{bmatrix} u_{dep} \\ v_{dep} \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{18}$$

where, $\{u_{dep}, v_{dep}\}$ and u,v are corresponding pixels in depth and RGB images respectively and elements a through f define the affine transformation between these 2 images. This equation can be solved as a linear system of the form $Ax = b$, whose unknown parameter x can be found by calculating the pseudo inverse $(A^T A)^{-1} A^T b$.

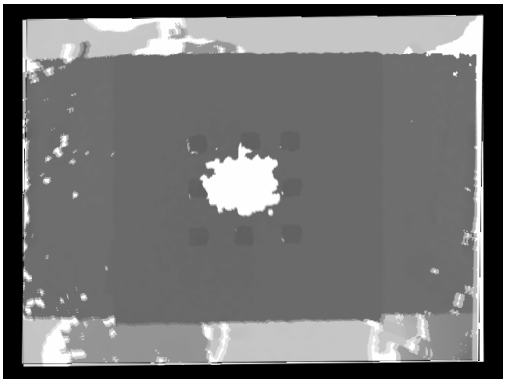


Fig. 4: The RGB and Depth images overlap after applying affine transformation

After aligning the RGB and the depth images, we can transform the pixel locations to camera frame coordinates by multiplying them with the camera intrinsic matrix. The camera intrinsic matrix was calculated by using the `camera_cal.py` script which captured numerous frames of a checkerboard in different poses. This matrix defines the intrinsic properties of the sensor such as, the focal lengths f_x, f_y and principal point offsets c_x, c_y. The calculated intrinsic matrix is given below:

$$K = \begin{bmatrix} 542.27975972 & 0.00 & 332.75615151 \\ 0.00 & 542.4745867 & 267.91209383 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{19}$$

The Kinect sensor was mounted overhead, facing down towards the Rexarm. This setup resulted in the camera and world planes parallel to each other and hence the real world depth of any point could be calculated using the following function of the 10-bit depth sensor data d,

$$z(u, v) = 12.36 + tan(d/2843.5 + 1.1863) \tag{20}$$

In addition, the parallelism between the camera and world planes simplified the problem of finding the relation between the pixel location and the world coordinates, into one of least squares fitting. The elements of the transformation matrix can be expressed as the solution to the equation,

$$\begin{bmatrix} u_1 & v_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_2 & v_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{n-1} & v_{n-1} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & u_n & v_n & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix} \tag{21}$$

Finally, we multiply transformation matrices obtained from Eq. (19), (20) and (20) to arrive at a real world coordinate X_w, Y_w from a pixel location u,v.

2) Block Detection: Our previous observation that the world plane and camera plane are parallel gave us an intuitive approach to block detection based on depth thresholding. In detail, the surface of any block on the world plane returns different depth sensor value relative to the Rexarm base. This approach also enabled us to distinguish and determine the height of different stacks of blocks. However, to avoid the issue of misclassifying parts of the Rexarm as a false positive, we only apply this process when the Rexarm is in the idle position. The algorithm is summarized as below:

Algorithm 1 Block Detector

[h!] **Inputs**: $DepthImage$
Outputs: $BlockContours$, $BlockCenterCoordinates$

```
Define Lower Depth Thresholds, lowThresh
Define Upper Depth Thresholds, highThresh
Define Area Thresh, areaThresh
for Each Depth Frame do
  for l,u in lowThresh,highThresh do
    Apply threshold [l,u] on each pixel
    Apply Open and Close filters
    Find Contours
    for Every countour in Range(areaThresh) do
      Find minimum enclosing rectangle
      Find moment of rectangle
      Update BlockContours
      Update BlockCenterCoordinates
    end for
  end for
end for
```

We began by defining a set of paired lower and upper depth threshold, $lowThresh$ and $highThresh$, which signified depth ranges from a single block to a vertical stack of five blocks. In addition, we also specified a

range within which the area of the contour of the block would lie, $areaThresh$, to reduce the possibility of classifying any non block contour as a false positive. Then, evaluated each pixel in the depth frame to check whether it lay in the bounded threshold, else it was assigned zero. The obtained mask was passed through an opening and closing operation to remove noise from the image and enhance the robustness of the detector. The processed mask was used to find the contour of detected objects whose area lay within the range specified by $areaThresh$. If any such contour was detected, we then found the set of corner points which defined the minimum enclosing rectangle for the specified contour and updated the list, $BlockContours$. We also used the mean of the corner points to calculate the center of the detected block and update $BlockCenterCoordinates$.

3) Color Identification: The idea behind our color identification process depends on the fact that different colors can be proposed as independent regions when depicted in a colour space. We selected to operate CIELAB (or Lab) color space as our choice. This color space allowed to define color thresholds, which we used to classify the different colors of the detected blocks by sampling the color space values at the $BlockCenterCoordinate$.

Algorithm 2 Color Identifier

Inputs: $BlockCenterCoordinates$
Outputs: $Colors$

```
for Each Image Frame do
    Define color thresholds, colThresh
    Convert from RGB to LAB
    for Each BlockCenterCoordinates do
        Sample LAB color value
        if colorvalue in Range(colThresh) then
            return Detected color
        else
            return No color detected
        end if
    end for
end for
```

E. Rexarm Motion Planning

Given the state of the gripper $s_g \in \{grasp, drop\}$, the task type $task \in \{grasp, drop\}$, the current configuration q_0, and the goal position in workspace q_f, the motion planning algorithm as shown in Algorithm 3 would return 0 if the task could not be performed or return 1, execute the planned trajectory and velocity profiles q_t, v_t, and perform the corresponding task.

Algorithm began checking if the state of the gripper conflicted with the task. If the state and task type were the same, then it was unnecessary to execute the task. Line 4 performs Inverse Kinematics (IK) as described in Section II-C2 for the goal position. The IK would determine if the goal position was reachable for the arm. If not, then the trajectory planner returned 0. Otherwise, the goal configurations are passed into Trajectory Smoothing function as described in Section II-B. The arm would then execute the generated smooth trajectory and velocity profiles q_t, v_t (Line 9). After reaching the goal position, the gripper opened to drop the object or closed to grasp the object (Line 10). Finally, the arm would return to its idle position and the idle position depended on the gripper state. If the gripper was open, the arm would go to the idle position pointing upwards. If the gripper has an object in hand, the arm would go to $q = [0, 0, 0, 0, \pi/4, 0]$ and toggle twice to adjust the block position as mentioned in Section II-A.

Algorithm 3 Motion Planner

Inputs: $q_0, p_f, s_g, task$

```
1:  if s_g == task then
2:      return 0
3:  end if
4:  q_f ← inverseKinematics(p_f)
5:  if q_f is None then
6:      return 0
7:  end if
8:  q_t, v_t ← trajSmooth(q_0, q_f, v_0, v_f, a_0, a_f, v_d)
9:  executePlan(q_t, v_t)
10: s_g ← toggleGripper(s_g)
11: returnToIdle(s_g)
12: return 1
```

The overall logic flow of the Rexarm system is shown in Figure 5. With all the tasks given, the "Block Detection" block implements the method discussed in Section II-D and provides the world coordinates, the orientations, and the colors of all the detected blocks. The "Prioritized Tasks" block prioritizes the tasks, specifies the tasks to grasp or drop, and provides the goal position of the gripper. If the task list is empty, end the process. The "Motion Planner" block implements Algorithm 3 which includes the gripper logic as discussed before. If the Rexarm executes the plan, the actual success or failure of the task is judged by the round block "Success/Failure". If failure is given or Motion Planner returns 0, adjusting the environment and/or resetting the arm is required. If task is successfully performed, go to the next task. If all the tasks are successfully performed, end the process.

III. Results

A. Gripper Performance

The performance of the gripper was evaluated by commanding it to pick and place blocks at different

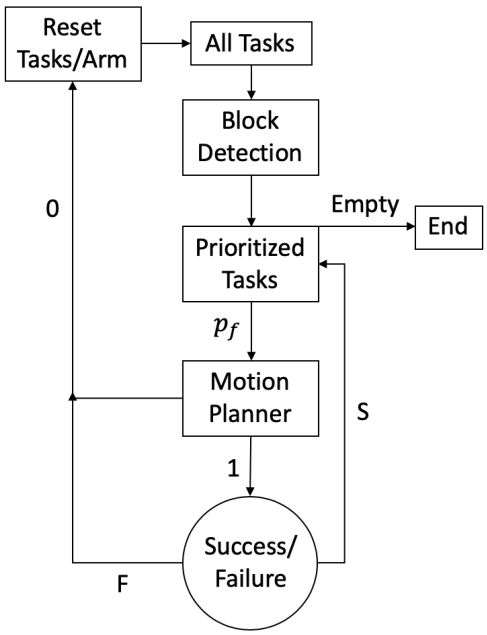


Fig. 5: Flow diagram of performing tasks

locations within the workspace of the robot arm. The performance evaluation for the gripper is shown in Table II and consists of two categories:

- Pick performance: Ability of the gripper to pick blocks with random orientations.
- Adjustment performance: Ability of the gripper to stabilize the block during intermediate pose after picking.

During the intermediate pose of the gripper, the arm is pointing upwards with leaned wrist and this configuration is defined by $q = [0, 0, 0, 0, \pi/4, 0]$. The placing performance of the gripper relies highly on block detection performance and inverse kinematics and will be discussed in further sections regarding the perception accuracy and competition performance.

Success Rate	Better Calibration (Error 0-1cm)	Worse Calibration (Error 1-2cm)
Pick	100%	90%
Adjustment	100%	80%

TABLE II: Gripper performance on picking and adjusting abilities under better and worse camera calibrations. Better calibration is represented by the error of the block position in the workspace.

It was observed that out of 10 trials, the gripper is able to pick up the block with a random orientation every time under better camera calibration and successfully correct the block pose. Under worse camera calibration, the gripper is only able to pick up the block 9 out 10 times and successfully stabilizes the block 8 times. Under worse camera calibration, the gripper could not reach the accurate block location and sometimes the supporting piece of the gripper might move the block during transition, which prevents the successful grasp. When the gripper just reaches its adjustment configuration and opens, despite the successful grasp, the block would fall down if its center is outside of the gripper's adjustment area. This occasionally happens when the estimated pose of the gripper is higher than the block pose. Improvement of the gripper performance will be mainly based on the improvement of camera calibration or removing the supporting piece which introduced challenges in block pickup.

B. Trajectory Smoothing

To analyze the performance of the quintic polynomial trajectory smoothing method, the trajectory created by the quintic polynomial is compared with the trajectory without smoothing at slow and fast speeds. The trajectory is divided into 3 segments:

1) from $q = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$ to $q = [1.0, 0.8, 1.0, 0.5, 1.0, 0.0]$
2) from $q = [1.0, 0.8, 1.0, 0.5, 1.0, 0.0]$ to $q = [-1.0, -0.8, -1.0, -0.5, -1.0, 0.0]$
3) from $q = [-1.0, -0.8, -1.0, -0.5, -1.0, 0.0]$ to $q = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$

and the quintic polynomial method is applied for each segment. The joint angles are collect at each time step with $\Delta t = 0.1s$ and the end-effector position is calculated by Forward Kinematics. The resultant trajectory comparison is shown in Figure 8 in Appendix B. At fast speed, $v_d = 1$ as defined in Section II-B, the quintic polynomial trajectory is more smooth than the one without smoothing. At low speed, $v_d = 0.5$, the the quintic polynomial trajectory is similar as the one without smoothing.

To visualize the performance of the trajectory smoothing, the comparison of quintic polynomial trajectories at fast and slow speeds is shown in Figure 6. Two trajectories at different speeds align with each other, which implies this trajectory smoothing method has stable performance at various speeds. To compare the smoothness of two trajectories, the velocity trajectories at fast speed with and without smoothing are shown in Figure 7. Without trajectory smoothing, the velocity of the end-effector suddenly increases at the beginning and decreases dramatically before stopping, which might throw the block away. In the contrary, with trajectory smoothing, the velocity of the end-effector gradually increases to the peak and gradually decreases to zero at the goal configuration. Trajectory smoothing would

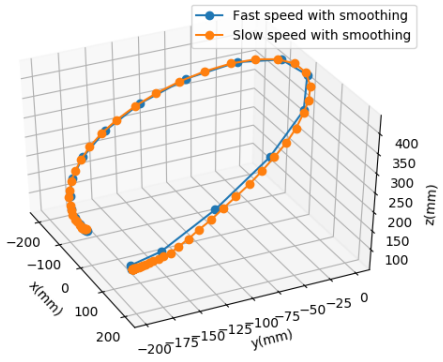


Fig. 6: Trajectory smoothing comparison between fast $v_d = 1$ and slow $v_d = 0.5$ speeds under the second segment

definitely provides more stable performance at fast speed and also limits the jerks of the arm and reduces the wear in the motors.

C. Tooltip Accuracy

The accuracy of forward kinematics to correctly specify the workspace coordinates was tested as per below process:

- Desired x, y coordinate of the tool tip was marked in the ground plane using vernier caliper
- The z coordinate of the tooltip was fixed by locating it either in the ground plane at the desired location or by locating the tool tip at the corner of a stack of blocks placed at the desired location
- The height of the blocks were measured using a vernier caliper.

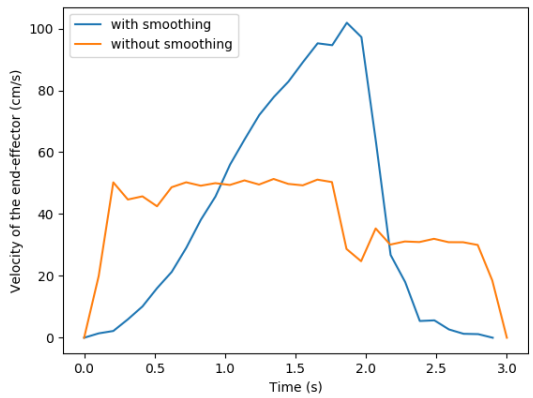


Fig. 7: Velocities of the end-effector at fast speed with and without path smoothing

- GUI was used to record the position of the tool tip as provided by the Forward Kinematics module
- Error was computed by subtracting the actual workspace coordinate from the desired workspace coordinate

To estimate the error correctly, the above process was repeated 3 times for each position. The table below summarizes the result from this study. Since the measurements were made with a vernier caliper, the below measurements have $\pm$ 0.05 mm error.

TABLE III: Tooltip Accuracy

Coordinates (x, y, z) in mm	**Error (mm)**
(290, 0, 0)	(2.74, 0.41, 0.62)
(0, -200, 0)	(0.63, 1.75, 0.55)
(-205, -145, 0)	(2.36, 1.38 0.47)
(215, 200, 77.12)	(2.55, 1.83, 1.16)
(-110, 120, 153.78)	(1.64, 1.25, 1.59)
(220, -215, 38.50)	(2.42, 2.02, 0.81)

The calculation of forward kinematic result requires the homogeneous transformation matrices which are based on the angle of the motor joints and the length of the link. Errors can be introduced based on the resolution of the encoder, lease count of vernier caliper and error in measurement. Additionally, errors in forward kinematics can also be attributed to the play in the joints which is not measured by the encoder.

D. Camera Calibration and Block Detection

In this section, we discuss the selection of points for finding the relation between the RGB and Depth images, the accuracy of the extrinsic calibration, the block detector and the color identifier.

1) Camera Calibration: We use the corner points of the workspace base of the Rexarm from RGB and corresponding corner points from the Depth Image to calculate the affine transformation. In addition, we use the same set of points from the RGB image and the manual measurement of corner points from the assumed world center to calculate the transformation matrix from pixel to real world coordinates. Over the course of 10 extrinsic calibrations with constant environment, we observe a mean error of 3mm with a standard deviation of $\pm$2mm along the X-axis and a mean error of 9mm with a standard deviation of $\pm$2mm along the Y-axis. For evaluating the depth perception, we first measure the distance of the workspace base from the Kinect sensor, which is found to be 950mm $\pm$5mm. We then subtract the depth measurement of the top of a block from the

base depth to measure the dimension of the block. The dimension of block evaluated by the Kinect is found to be close to the real world measurement of the block $\pm$38mm within a range of 1-2mm.

2) Block Detector: The lower and upper thresholds provided to Algorithm 1 are found through trial and error and have 100% accuracy during the course of the project. Furthermore, we specify the acceptable contour area range, $areaThresh = [300, 900]$. We also observe that proposed block center might not coincide with the real block center due to noisiness of the image data. We place blocks at points along squares of dimensions 100mm $\pm$2mm and 200 $\pm$2mm and compare the proposed block centers with the block centers to evaluate the error in measurement which are plotted in Fig 9 in Appendix D.

3) Color Identifier: We originally proposed color thresholding based on the HSV color space. However, the variance in light intensity results in change of Saturation values of the pixel in an image. To counter this problem, we decided to operate in the Lab color space which is much more robust with respect to change in lighting conditions. We were successful in identifying the colors Orange, Blue, Green and Purple with an accuracy of 86% of the cases. The colors Red and Pink were misclassified as one another, due to the small separation between the threshold values, with a low accuracy of 60% and 56%. While Lab color space facilitates easy separation of different colors, it is still not completely light intensity invariant. This became apparent in the case of Yellow and Black, where the change in lighting conditions would shift threshold margins by a large amount resulting in an accuracy of just 76% for both the colors.

E. Competition Performance

The performance of the robot for different tasks in the competition are evaluated below.

Task 1 of the competition required the arm to stack three blocks as fast as possible. The robot arm took more than 1 minute to complete the task. The arm was not fast enough because of introduction of many intermediate states introduced to pick and place the blocks. Also, due to trajectory smoothing we spend additional time completing movements at slower speeds when the gripper does not have the block in hand.

Task 2 of the competition required the arm to line up 8 blocks as per a specified color order. The robot arm lined up 3 block in the color order. The performance of the arm can be attributed to poor color detection of blocks and due to limited workspace of the robot.

Task 3 required the arm to stack up 8 blocks as per a specified color order. In this case, the arm stacked up three blocks in the color order. Besides inaccurate color detection, our robot arm had limited ability to stack more than 3 blocks because of fixing α as $0\,\mathrm{deg}$.

Although task 4 and task 5 were not attempted due to limited availability of time, task 4 could be achieved by teach and repeat function with accurate trajectory planning. Task 5 is relatively complicated since it required the detection on tilted plane and an additional frame transformation.

IV. Discussion

There is much scope for improvement in the design and implementation of the robot arm. Firstly, the motion plan of the rexarm can be improved to increase the speed during phases of picking the block. Secondly, to expand the workspace of the robot, inverse kinematics could be improved to determine the angle of approach α while picking and dropping blocks. Thirdly, to improve the ability to detect different colors, attempts could be made to ensure a uniform light environment during operation phase (by making use of cardboard to block external light). Fourthly, the current calibration procedures introduce errors which could be reduces by implementing April tags.

V. Conclusion

In conclusion, a 6 DOF robot arm was designed and implemented for executing various tasks. This implementation required integration of several core areas of robotics such as kinematics, perception, motion-planning and reasoning. The performance of the arm in the competition was evaluated and improvements were identified for future.

References

[1] "Rob 550 arm lab manual."

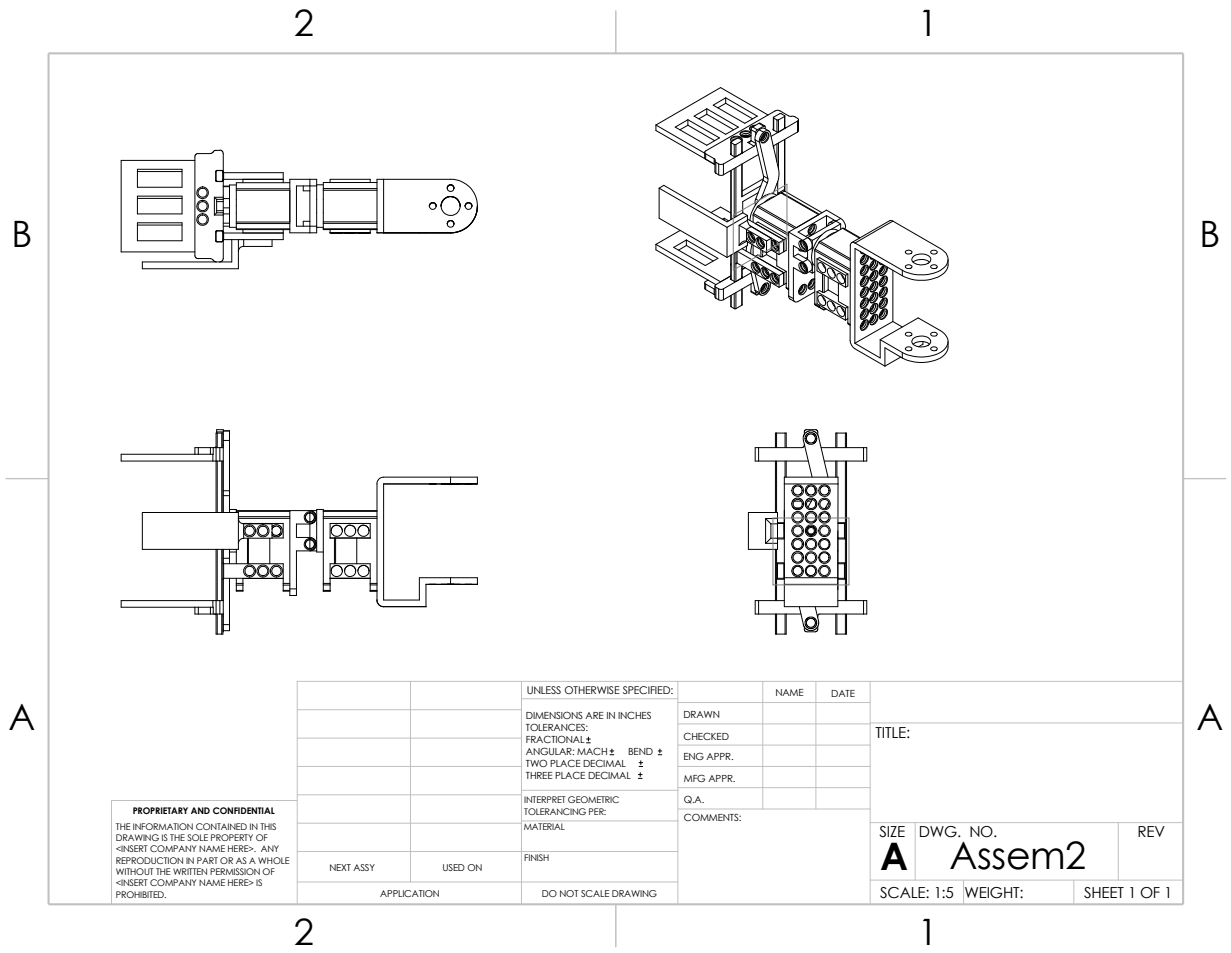
[2] M. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*. Wiley, 2005. [Online]. Available: https://books.google.com/books?id=wGapQAAACAAJ

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

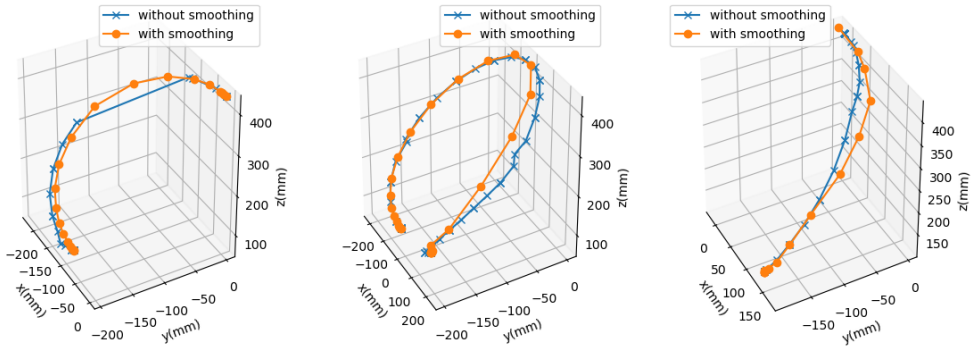
[4] S. E. Macfarlane and E. A. Croft, "Jerk-bounded manipulator trajectory planning: design for real-time applications," *IEEE Trans. Robotics and Automation*, vol. 19, pp. 42–52, 2003.

VI. Appendices

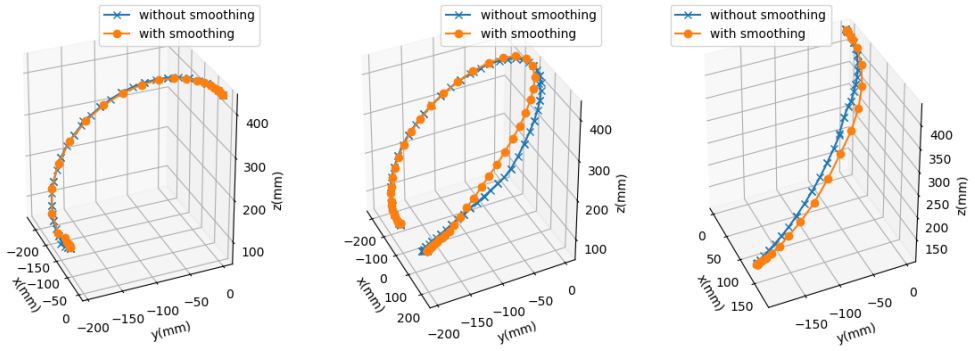
A. Gripper Drawing



B. *Trajectory smoothing*



(a) Fast speed $v_d = 1$



(b) Slow speed $v_d = 0.5$

Fig. 8: Trajectories with and without path smoothing under fast $v_d = 1$ (a) and slow $v_d = 0.5$ (b) speeds in three segments from left to right

C. Kinematics

TABLE IV: Inverse Kinematic Symbol Definition

Symbol	Definition
l_1	Length of link from ground to shoulder joint
l_2	Length of link from shoulder joint to elbow joint
l_3	Length of link from elbow joint to wrist
l_4	Length of link from wrist to end effector
θ_1	Rotation angle of base joint
θ_2	Rotation angle of shoulder joint
θ_3	Rotation angle of elbow joint
α	Rotation angle of gripper relative to ground
d	Distance of end effector from base frame in XY plane

D. Perception System

TABLE V: Depth thresholds for block detection

Stack Level	Lower Threshold	Upper Threshold
1	175	177
2	170	175
3	165	170
4	160	165
5	155	160

TABLE VI: Color thresholds for color identification

Color	L Range	a Range	b Range
Yellow	220-255	170-135	100-140
Black	30-110	125-140	110-135
Orange	70-130	150-180	40-70
Blue	100-155	140-165	130-150
Green	130-155	110-125	125-140
Red	55-110	160-185	60-80
Pink	95-140	150-200	90-100
Purple	85-135	140165	105-125

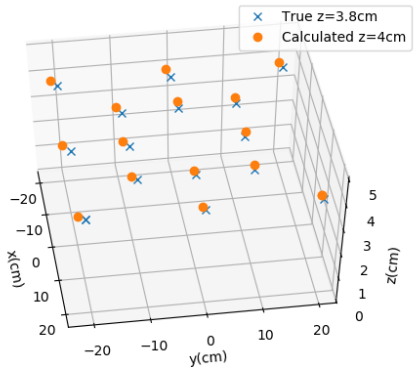
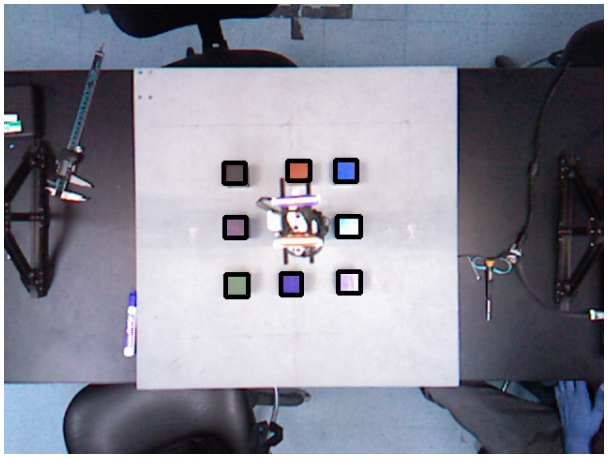
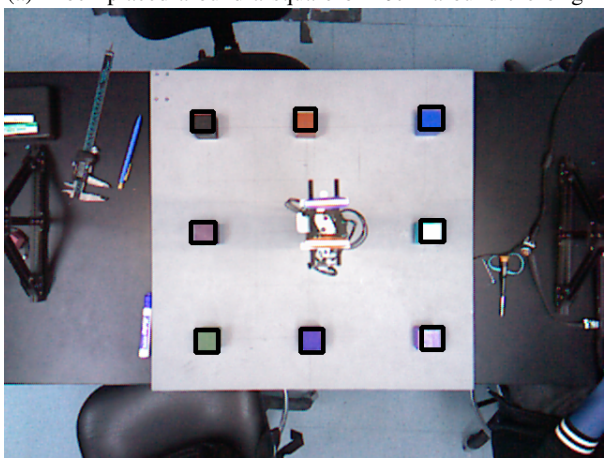


Fig. 9: Comparison of Actual block coordinates v/s Calculated block coordinates



(a) Block placed around a square of 10cm around the origin



(b) Block placed around a square of 20cm around the origin

Fig. 10: Setup for evaluating extrinsic calibration and block detector accuracy