Final Report for Introduction to Robotics Lab Arm Project

Aatiqa Khalid, Syed Mujtaba Hassan, and Syeda Manahil Wasti
Habib University, Pakistan
Email: ak06889@std.habib.edu.pk, ms06948@std.habib.edu.pk, sw06877@std.habib.edu.pk

***Abstract*—Our project aims to develop an efficient arm that can perform a range of pick and place tasks effectively. In this project we explored the Phantom X Pincher arm along with its features. We focused on learning about the arm's capabilities and enhancing them by incorporating visual sensing through Intel RealSense SR305 camera and making it perform different pick and place tasks. The project is of importance especially in today's modern age where the demand for precise and reliable automation is on the rise. Robot arms are integrated into automated systems and the overall performance relies on how well they execute their specific tasks. Our quantitative results showed that the arm had good pose accuracy and repeatability for pick and place tasks. The arm also had reliable object detection using vision based sensing as well as when given coordinates of the objects. Qualitatively, we gained insight into the arm's hardware, kinematics, control, and singularity analysis in various pick and place tasks. In the future, we can look into path planning algorithms to enhance the pick and place tasks by obstacle avoidance (such as potential field algorithms etc.). Another key area to explore can be sensor fusion as data from different sensors can be integrated to understand the surroundings better, and to perform more complex tasks.**

***Index Terms*—Phantom X Pincher, Intel RealSense SR305, Pick and Place, Kinematics, Control, Singularity Analysis, Path Planning, Potential Field Algorithms, Sensor Fusion**

I. Introduction

We used the Phantom X Pincher arm in this project:

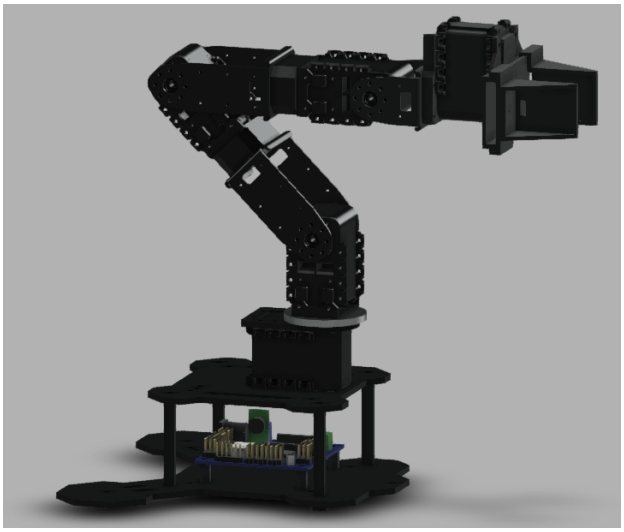


Fig. 1: Phantom X Pincher arm robot [1]

The main goal of our arm project was to perform pick and place tasks of various complexity such as sorting different cubes based on color, or picking a cube from a known position and placing at a provided position in workspace of the robot. We performed these tasks using different methods including visual-based sensing using an Intel RealSense SR305 camera:

Fig. 2: Intel RealSense SR305 camera [2]

The use of the depth camera makes the pick and place tasks faster and more accurate. Pick and place robots have a lot of advantages such as higher productivity, greater accuracy, improved worker safety, cost-effectiveness, flexibility, and the ability to gather and analyze operational data. These advantages make them an appealing choice for automating repetitive tasks in industries.

Our project's aim was to make the arm successfully execute pick and place tasks autonomously using position-based visual servoing[1]. The significance of this project is particularly relevant in today's era, where the need for accurate and dependable automation is increasing. Robotic arms play a vital role in automated systems, and their overall performance is dependent on their proficiency in carrying out specific tasks. The arm can be integrated in an automated system to perform even more complex functions [3].

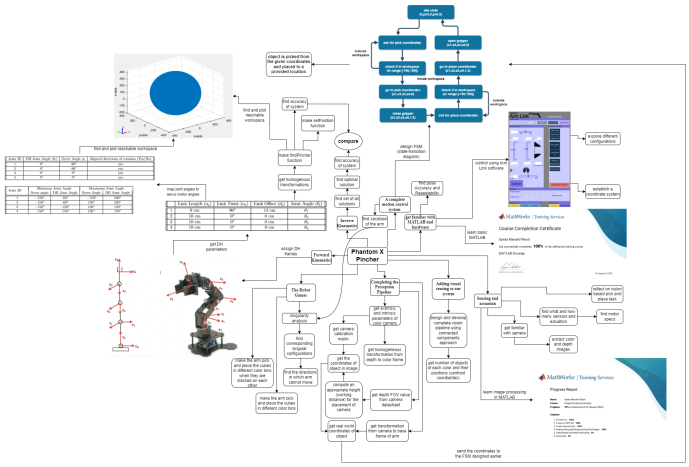
This report is divided into 3 main sections. First is the Methodology section. This will have subsections for each of the modules that we worked on for the project. The second one is the Results section where we discuss the quantitative and qualitative results for the performance of our system. In the

[1]position-based visual servoing: coordinates of positions of interest are determined from the image and the robot motion controller moves the arm to desired positions.

end, the third section is the Conclusion in which we summarize the final results of our project, discuss what we learned from this project, and what we would do differently to enhance the performance.

II. Methodology

A block diagram of our complete robotic system is given below (the image is available in the GitHub repository and can be viewed more clearly from there):



A. Getting familiar with MATLAB and the hardware

The goal of this module was to learn basic MATLAB as the project code was done using that, and also to get familiar with the Phantom X Pincher hardware and its control using ArmLink software. We spent the first lab in learning basic MATLAB. In the second lab we were getting familiarized with the arm. We found the number of joints and links and also calculated the degrees of freedom of the manipulator using Grubler's formula:

$$\text{Degrees of Freedom} = m(N-1) - \sum_{i=1}^{J} c_i = m(N-1-J) + \sum_{i=1}^{J} f_i$$

Fig. 3: Grubler's formula

After that we explored the capabilities and limitations of this arm using the ArmLink software and checking different configurations. For example the limit the arm can reach, extended in one direction.
Lastly, we designed and conducted an experiment to find out the pose accuracy and repeatability of the arm to see how it will perform when doing pick and place tasks. We made the the robot arm hold a pen then made the arm mark 5 different points in cartesian coordinates. Then we re-entered the points and returned the arm to each of the points 3 times. After that we found the difference between the 3 marks of each point. We used the following formula to calculate the position pose accuracy:

$$AP_P = \sqrt{(\bar{x}-x_c)^2 + (\bar{y}-y_c)^2 + (\bar{z}-z_c)^2}$$

Fig. 4: Position Pose Accuracy [4]

For pose repeatability we used the formula given below:

$$RP_l = \bar{l} + 3S_l,$$

where

$$\bar{l} = \frac{1}{n}\sum_{j=1}^{n} l_j$$

$$l_j = \sqrt{(x_j-\bar{x})^2 + (y_j-\bar{y})^2 + (z_j-\bar{z})^2}$$

$$S_l = \sqrt{\frac{\sum_{j=1}^{n}(l_j-\bar{l})^2}{n-1}},$$

Fig. 5: Pose repeatability [4]

B. Sensing and actuation for the arm

We identified the different sensors and actuators in the arm and the specifications for the motor used in it. We also got familiar with using the Intel RealSense SR305 camera and completed 4 modules of the Image Processing Onramp course. After that we extracted the depth and color images that we got from the camera using MATLAB functions depth_example() and get_color_frame().

C. Forward Kinematics

This part of our project was to familiarize us with the forward kinematics of the arm. We used DH convention to assign frames to the arm and found out the DH parameters. We found the intermediate and resultant homogeneous transformations from the arm's base to end-effector frame.
After that we used these transformations to make a MATLAB function function [x, R] = findPincher(theta_1, theta_2, theta_3, theta_4) that accepts joint angles of Phantom X Pincher and returns the end-effector position and orientation in the specified order. We also used the pincherModel.m file to enter joint angles for different configurations and displayed a skeleton of the robot with frames at the corresponding position and orientation.
Next we mapped the DH joint angles to the respective servomotor angle. We determined the angular shift between each joint angle and the joint position when 0° command is sent to the corresponding servomotor, and we also checked if the positive directions of rotation in the two cases are aligned or not. Using these angular shifts we found the joint limits.
Then we used those limits to identify the reachable workspace and plotted it in MATLAB. After that we made a MATLAB function errorCode = setPosition(jointAngles) that takes joint angles as inputs and and sets them as goal positions for the respective motors in the arm. In the end we verified the positioning by selecting 5 random joint configurations for the manipulator and determining the end-effector position and

orientation from findPincher function. We then used our setPosition function to move the end-effector to each of the five chosen configurations and for each configuration, physically measured and noted the actual end-effector position. Lastly, we computed the error using Euclidean distance between the position computed by our function and the actual position achieved by the arm.

D. Inverse Kinematics

In this module we derive and implement a solution to the inverse kinematics problem for the Phantom X Pincher. We find the joint parameters given the end-effector position and orientation [4].
We were given the position (x, y, z) and orientation ϕ. Using these, we found the mathematical expressions for all inverse kinematics solutions as well as number of solutions.
After that we wrote a MATLAB function findJointAngles(x,y,z,phi) that accepts position and orientation of the end-effector as input and returns all IK solutions in matrix form.
Then we wrote another MATLAB function findOptimalSolution(x,y,z,phi) to find an optimal solution from the matrix containing all possible solutions. The optimal solution is the one in which there is minimum change in the joint angles i.e., there is minimum movement of arm from its current configuration. We also checked if the solution was realizable i.e., the angles were within specified limits such that the arm doesn't self-collide or stop moving at some point that is outside the workspace.
In the end we determined the accuracy of our system and compared it to the value obtained in previous module. We selected five points (x, y, z, ϕ) in the workspace and used our function findOptimalSolution to find the point that the robot reached. Then we measured the point physically and used Euclidean distance to calculate the error.

E. A complete motion control system

In this module we designed an FSM to complete a pick and place task based on the pick and place position coordinates which were given as input. Then implemented the FSM by coding on MATLAB A summary of our strategy is given below:
Explanation of strategy:
The program starts with the arm in an idle position and an open gripper. The user can choose to execute pick and place actions or remain in the idle state.
For the pick action, the user provides joint angles for the pick position. If the coordinates are valid, the arm moves to that position, closes the gripper to hold the object, and returns to the idle position. If the coordinates are invalid, the user is prompted for input again.
Next, the user provides joint angles for the place position. If the coordinates are within the arm's workspace, the arm moves to that location, opens the gripper, and then raises to a certain height before returning to the idle state. This prevents displacement of the object. If the place coordinates are not valid, the user is asked for input again.
We also determined the Jacobian of the arm in this module by using the e DH parameters and homogeneous transformation from base to end-effector frame obtained in the previous modules.

F. Adding visual sensing to our system

In this section we determine the input coordinates for the motion control system that we designed earlier by using visual sensing through Intel RealSense SR-305 camera to detect the objects (cubes).
We designed and developed vision pipeline to determine the position and color of the cubes that are to be picked and placed.
We used the connected components [5] approach to segment the image we get from the camera.[2]
In this method we identify and group together pixels or regions in an image that belong to the same object i.e., we identify objects in an image by analyzing the connectivity between adjacent pixels or regions. The connectivity of a connected component describes how its pixels or image elements are connected to each other.
We used connectivity of 8 for the 2-D images. Two pixels are considered 8-connected if they are adjacent horizontally, vertically, or diagonally. In other words, the pixels can share a common side or a common corner. A short and simple pseudo-code of our approach is provided in Appendix A (all the detail, explanation and commented MATLAB code is given in the GitHub repository).

G. Completing the Perception Pipeline

This module was the one in which we combined our previous modules to make the arm execute pick and place task with visual sensing using image from the Intel RealSense SR-305 camera.
We first find the intrinsic and extrinsic parameters from the given functions determineIntrinsics and determineExtrinsics.
We also determined the values of depth field of view, depth start point, resolution, and frame rates for both cameras i.e., color and IR.
Then we used depth field of view to calculate the height at which camera should be so that the entire workspace is visible. Using this height we get the transformation from camera to robot base frame. Then we use these values to get the real world coordinates of objects from image coordinates that we got from previous module (centroids of the objects).
Lastly, we gave these points as input to the FSM designed earlier and performed vision-based pick and place.

H. The Robot Games

This was the last module we worked on where we had to make the arm execute complex pick and place task. We first did singularity analysis by finding where the determinant of the Jacobian we found in module 6 becomes 0 so that the arm can avoid the configurations where singularity occurs.

[2]Segmentation is the process of collecting together pixels into summary representations that emphasize some important, interesting, or distinctive properties, e.g. pixels of the same color [4]

Our strategy was to use the code from module 7 to get the image coordinates of objects and then transform them to real world coordinates by using the code from module 8. After that we will give the obtained real world coordinates as input to the FSM we designed in module 6.
We mark 2 spots in our workspace for red and yellow bin. From the code of module 7 we also get the color of each object and the number of objects of each color. We will use this information to pick and place the objects into respective bins according to their colors.
After that we had to perform pick and place task that also involved stacked cubes. For this, the distance of camera from the objects will change rather than stay at 805mm (height of camera from base of arm calculated using depth FOV we got from datasheet of Intel RealSense SR-305 camera) so we will make this adjustment in our previous task's code and make the z coordinate of cube in world coordinates, a variable. The rest of the pick and place task remains the same.

III. Results

A. Getting familiar with MATLAB and the hardware

The arm has 4 joints 5 links as shown below:

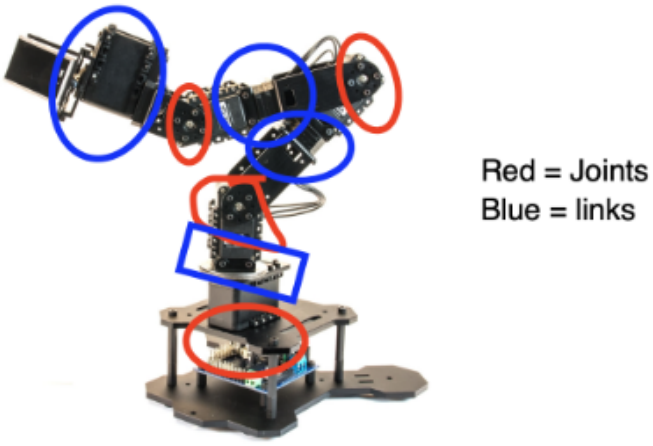


Fig. 6: Joints and Links

All the joints are revolute:

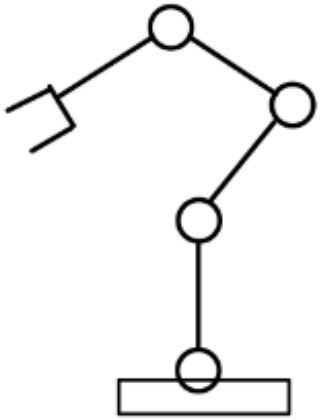


Fig. 7: Symbolic representation of the kinematic chain

After that we found the degrees of freedom, which was 4. We found that the arm can reach the farthest in an upright position as shown:

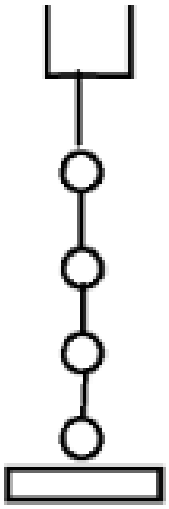


Fig. 8: Configuration in which the arm reaches the farthest possible point

Using ArmLink software we found that the wrist angle had like a human wrist motion but to a certain limit. At different points, changing the wrist angle affected the position of different links and joints. We used the software to move the arm and perform pick and place task . After that we found the directions of positive x, y, z axes and that 1 unit in ArmLink corresponds to 1mm in real world.
For pose accuracy and repeatability we found that the accuracy decreased as the distance between the base and the gripper increased which makes sense as the arm wobbles more as it stretches. Also we realized that if both accuracy and repeatability of our robot arm is lower, then the pick and place pipeline will be adversely affected and the arm will not be able to perform the task correctly. Our experiment had good

repeatability and bad accuracy, which resulted in deviation of position of the arm from the given points.

B. Sensing and actuation for the arm

We found that the arm has a potentiometer used as a position sensor and has 5 Dynamixel AX-12A servo motors (one for each of the 4 joints and one for the gripper) that are the actuators. The motor specifications we got from the datasheet were:

1) **Angle rotation limits:** CCW= 150° - 300°
   CW= 0°- 150°
2) **Resolution:** 0.29°
3) **Speed limit:** In Joint mode, max limit is 114 rpm.
   No Load Speed: 33.2 rpm
4) **Torque limit:** Stall torque: 1.5 [N.m] (at 12 [V], 1.5 [A])

We also found that the motor's resolution will limit the possible Cartesian resolution of the end-effector because if motor has limited resolution then it can move only in a limited number of positions and the robot will not have a greater volume of space to operate in. Thus, the Cartesian resolution will also be limited.
The extracted depth and color frames we got from the camera using MATLAB functions depth_example() and get_color_frame() were:

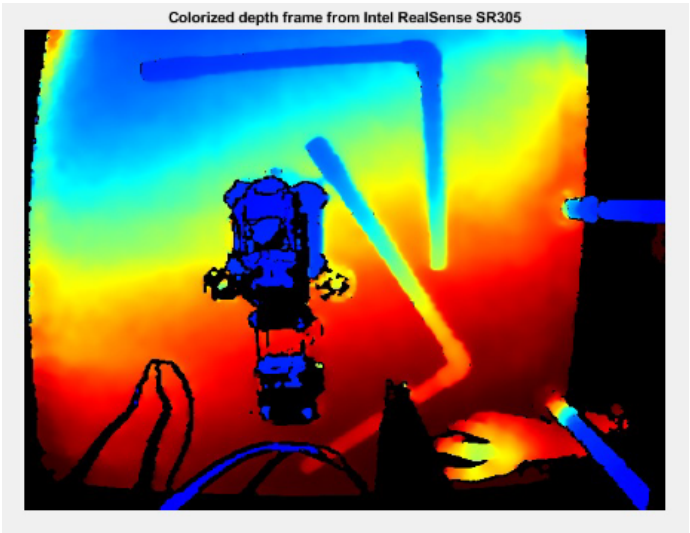


Fig. 9: extracted depth image

Fig. 10: extracted color frame

Highly repeatable processes are used by vision-based pick-and-place robotic systems. In many pick-and-place applications, the arm must not only determine the object's location and orientation, but also its structure, as well as the optimum way to choose or place it. We observed that in our case, the arm had good repeatability. The sensor and the vision system will involve the camera in our case which will be used to identify the object and perform the pick and place task.

C. Forward Kinematics

We assigned the DH frames as shown below:

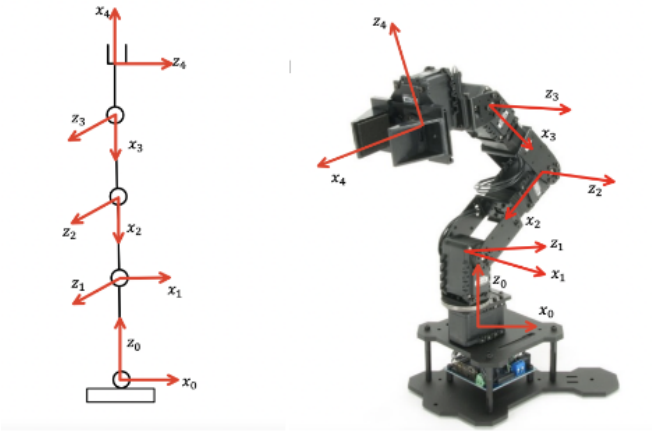


Fig. 11: DH frames assignment

The DH parameters were found to be:

	Link Length (a_i)	Link Twist (α_i)	Link Offset (d_i)	Joint Angle (θ_i)
1	0 cm	90°	13 cm	θ_1
2	10 cm	0°	0 cm	θ_2
3	10 cm	0°	0 cm	θ_3
4	10 cm	0°	0 cm	θ_4

Fig. 12: Table of DH parameters

One of the configurations we got using picherModel.m is given below:

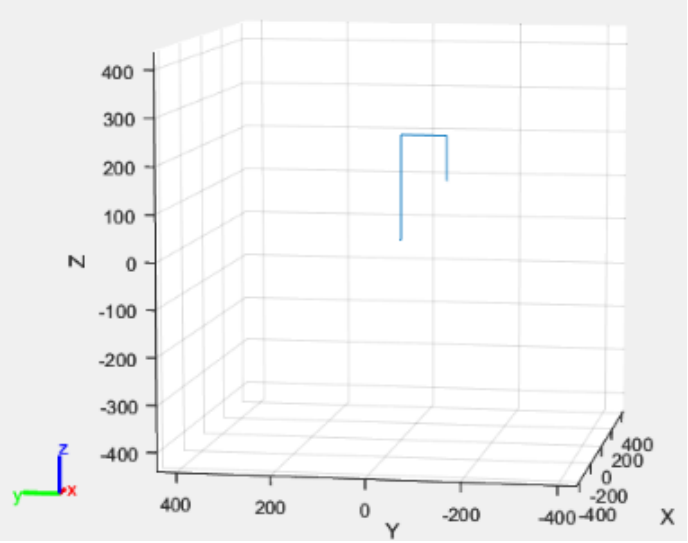


Fig. 13: configuration for joint angles $(\frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2}, \frac{\pi}{2})$

We saw that the function returns the same position and orientation for different configurations which we saw our arm in for the same joint angles by using the setPosition function. We found the following angular shifts between the servo motor and joint angles:

Joint ID	DH Joint Angle (θ_i)	Servo Angle ψ_i	Aligned directions of rotation (Yes/No)
1	0°	-90°	yes
2	0°	-90°	yes
3	0°	0°	yes
4	0°	0°	yes

Fig. 14: angular shift between servo motor and joint angles

We then used these angular shifts to map the servo motor angles to the joint angles as follows:

Joint ID	Minimum Joint Angle		Maximum Joint Angle	
	Servo angle	DH Joint Angle	Servo Angle	DH Joint Angle
1	−150°	-60°	−150°	240°
2	−150°	-60°	150°	240°
3	−150°	-150°	150°	150°
4	−150°	-150°	150°	150°

Fig. 15: servo motor angles mapped to the joint angles

The reachable workspace we plotted in MATLAB was:

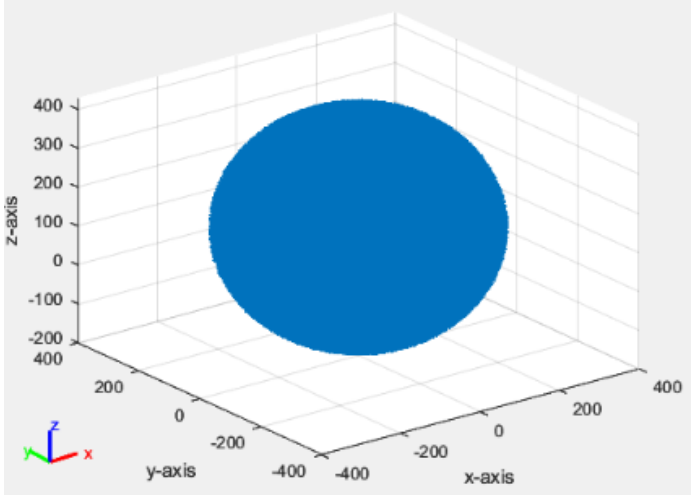


Fig. 16: Reachable workspace of the arm

As seen from the figures attached above, the maximum horizontal reach is -300mm to 300mm.
Lastly, we calculated the error between position computed by the findPincher function and the actual position our arm reached using setPosition function. We found that there was little to no error in the x, y, z values we got from both the functions.

D. Inverse Kinematics

Using inverse kinematics we found the following set of solutions:

First set of solutions

$$\theta_{11} = \tan^{-1}\left(\frac{y}{x}\right)$$
$$\theta_{21} = \pi - \beta$$
$$\theta_{31} = \gamma - \alpha$$
$$\theta_{41} = \phi - \theta_{21} - \theta_{31}$$

Second set of solutions

$$\theta_{12} = \tan^{-1}\left(\frac{y}{x}\right)$$
$$\theta_{22} = \beta - \pi$$
$$\theta_{32} = \gamma + \alpha$$
$$\theta_{42} = \phi - \theta_{22} - \theta_{32}$$

Third set of solutions

$$\theta_{13} = \pi + \tan^{-1}\left(\frac{y}{x}\right)$$
$$\theta_{23} = \beta - \pi$$
$$\theta_{33} = \pi - \gamma + \alpha$$
$$\theta_{43} = \phi - \theta_{23} - \theta_{33}$$

Fourth set of solutions

$$\theta_{14} = \pi + \tan^{-1}\left(\frac{y}{x}\right)$$

$$\theta_{24} = \pi - \beta$$

$$\theta_{34} = \pi - \gamma - \alpha$$

$$\theta_{44} = \phi - \theta_{24} - \theta_{34}$$

Where

$$\alpha = \cos^{-1}\left(\frac{a_2^2 - a_3^2 + (\sqrt{y^2+x^2} - a_4\sin\phi)^2 + (z - d_1 - a_4\sin\phi)^2}{2a_2\sqrt{(\sqrt{y^2+x^2} - a_4\sin\phi)^2 + (z - d_1 - a_4\sin\phi)^2}}\right)$$

$$\beta = \cos^{-1}\left(\frac{a_2^2 + a_3^2 - (\sqrt{x^2+y^2} - a_4\cos\phi)^2 - (z - d_1 - a_4\sin\phi)^2}{2a_2a_3}\right)$$

$$\gamma = \tan^{-1}\left(\frac{z - d_1 - a_4\sin\phi}{\sqrt{x^2+y^2} - a_4\sin\phi}\right)$$

and

$$a_2 = a_3 = a_4 = 10 \text{ cm}$$

$$d_1 = 13 \text{ cm}$$

$$\phi = \theta_2 + \theta_3 + \theta_4$$

The functions we made can be found in the GitHub repository. It has all the details along with commented MATLAB code to explain the approach.
In the end we calculated the accuracy of our system which turned out to be less than the one we computed with forward kinematics. We also found that there are points in the workspace for which all possible solutions are realizable meaning no singularity occurs i.e., there are no infinite solutions possible.
We can use the findJointAngles function to find solutions for different points and then use findPincher function to check if all 4 of the solutions give back the same point or not. If yes, then the point is one for which all possible solutions are realizable.

E. A complete motion control system

For the designed FSM to perform a pick and place task, we executed our code by using the following coordinates (joint angles):
pick position: [0,pi/3,1.27409,0]
place position: [pi/2,pi/3,1.27409,0]
Video of best execution:
https://youtu.be/ZY2zAZKbfW8
We found that the system performed well and was successful in completing the pick and place operation. However we had some points of improvement:
Points of improvement:

- We are currently taking direct input of joint angles. The code can be modified to take cartesian coordinates and then those can be converted to joint angles for a better execution.
- We have hard-coded the gripper value for the object we are picking i.e., the block. However, for different objects, gripper value should be variable.
- We have used 65 as the speed whereas it can also be taken as input from user so the arm can move at variable speeds.

The Jacobian we found for the arm is given below:

$$\begin{pmatrix} 10\cos(\theta_4)\,\sigma_{13} - 10\cos(\theta_2)\sin(\theta_1) + 10\sin(\theta_4)\,\sigma_{14} + 10\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) - 10\cos(\theta_2)\cos(\theta_3)\sin(\theta_1) & \sigma_{11} - \sigma_{12} - 10\cos(\theta_1)\sin(\theta_2) - \sigma_6 - \sigma_5 & \sigma_{11} - \sigma_{12} - \sigma_6 - \sigma_5 & \sigma_{11} - \sigma_{12} \\ 10\cos(\theta_1)\cos(\theta_2) - 10\cos(\theta_4)\,\sigma_{15} - 10\sin(\theta_4)\,\sigma_{16} - 10\cos(\theta_1)\sin(\theta_2)\sin(\theta_3) + 10\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) & \sigma_9 - \sigma_{10} - 10\sin(\theta_1)\sin(\theta_2) - \sigma_4 - \sigma_3 & \sigma_9 - \sigma_{10} - \sigma_4 - \sigma_3 & \sigma_9 - \sigma_{10} \\ 0 & 10\cos(\theta_2) + \sigma_8 - \sigma_7 + \sigma_2 - \sigma_1 & \sigma_8 - \sigma_7 + \sigma_2 - \sigma_1 & \sigma_2 - \sigma_1 \\ 0 & \sin(\theta_1) & \sin(\theta_1) & \sin(\theta_1) \\ 0 & -\cos(\theta_1) & -\cos(\theta_1) & -\cos(\theta_1) \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

where:

where

$$\sigma_1 = 10\sin(\theta_4)\,(\cos(\theta_2)\sin(\theta_3) + \cos(\theta_3)\sin(\theta_2))$$

$$\sigma_2 = 10\cos(\theta_4)\,(\cos(\theta_2)\cos(\theta_3) - \sin(\theta_2)\sin(\theta_3))$$

$$\sigma_3 = 10\cos(\theta_3)\sin(\theta_1)\sin(\theta_2)$$

$$\sigma_4 = 10\cos(\theta_2)\sin(\theta_1)\sin(\theta_3)$$

$$\sigma_5 = 10\cos(\theta_1)\cos(\theta_3)\sin(\theta_2)$$

$$\sigma_6 = 10\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)$$

$$\sigma_7 = 10\sin(\theta_2)\sin(\theta_3)$$

$$\sigma_8 = 10\cos(\theta_2)\cos(\theta_3)$$

$$\sigma_9 = 10\sin(\theta_4)\,\sigma_{13}$$

$$\sigma_{10} = 10\cos(\theta_4)\,\sigma_{14}$$

$$\sigma_{11} = 10\sin(\theta_4)\,\sigma_{15}$$

$$\sigma_{12} = 10\cos(\theta_4)\,\sigma_{16}$$

$$\sigma_{13} = \sin(\theta_1)\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3)\sin(\theta_1)$$

$$\sigma_{14} = \cos(\theta_2)\sin(\theta_1)\sin(\theta_3) + \cos(\theta_3)\sin(\theta_1)\sin(\theta_2)$$

$$\sigma_{15} = \cos(\theta_1)\sin(\theta_2)\sin(\theta_3) - \cos(\theta_1)\cos(\theta_2)\cos(\theta_3)$$

$$\sigma_{16} = \cos(\theta_1)\cos(\theta_2)\sin(\theta_3) + \cos(\theta_1)\cos(\theta_3)\sin(\theta_2)$$

F. Adding visual sensing to our system

Our designed FSM state transition diagram is given below:

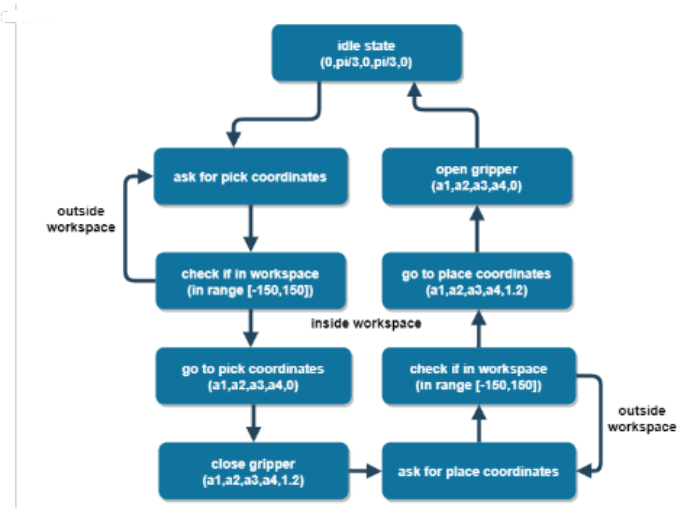


Fig. 17: state-transition diagram of designed FSM

We were successful in finding the number objects of each colour from the image that we got from the Intel RealSense SR-305 camera and we also found the centroids of all objects.

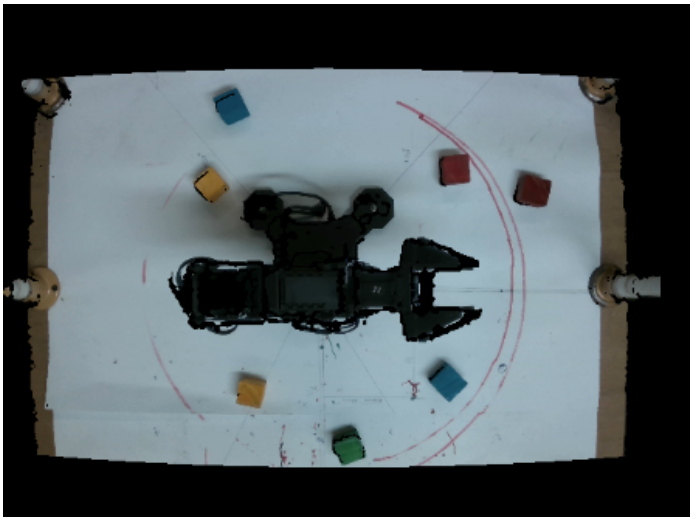


Fig. 18: color image from camera

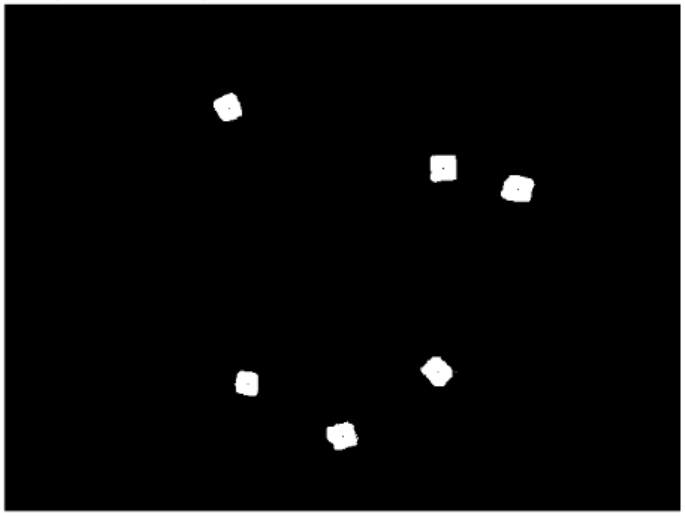


Fig. 19: cubes with only their top face visible and their centroids marked in black

```
num_of_obj = 6
```

```
red_obj = 2
green_obj = 1
blue_obj = 2
yellow_obj = 1
```

Fig. 20: output with the number of objects of each color

We can see that all the objects are detected perfectly along with their colors except for one yellow cube which is not even detected as it is placed too close to the robot.
This will now help us in picking and placing the objects i.e., the cubes present in the workspace without providing their locations to our system.

G. Completing the Perception Pipeline

We got the intrinsic parameters of the color and IR cameras using the determineIntrinsics() function as follows:

```
color_intrinsics =

  struct with fields:

      width: 1920
     height: 1080
        ppx: 952.0654
        ppy: 538.3796
         fx: 1.4092e+03
         fy: 1.4092e+03
      model: 0
     coeffs: [0 0 0 0 0]
```

Fig. 21: intrinsic parameters of the color camera

```
depth_intrinsics =

  struct with fields:

     width: 640
    height: 480
       ppx: 313.1847
       ppy: 245.7956
        fx: 475.7621
        fy: 475.7621
     model: 2
    coeffs: [0.1256 0.1459 0.0051 0.0066 -0.0759]
```

Fig. 22: intrinsic parameters of depth camera

We get the camera calibration matrix for depth camera as follows:

```
K = 3×3
10^3 ×
    1.4089         0    0.9507
         0    1.4089    0.5331
         0         0    0.0010
```

Fig. 23: camera calibration matrix of depth camera

We found the optimum height of the camera so that the entire workspace is visible to be 805mm.

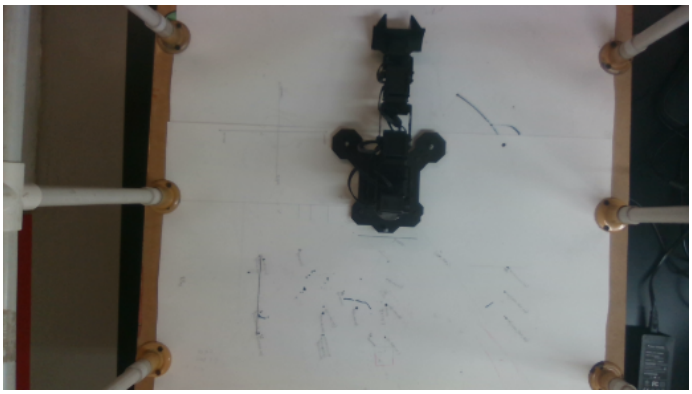


Fig. 24: view of entire workspace from camera height of 805mm

After that we find transformation of camera frame to robot's base frame by rotating it by 180° about x-axis and translating by (35, 0, 805). We get the following matrix

$$T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(180°) & -sin(180°) \\ 0 & sin(180°) & cos(180°) \end{bmatrix}$$

Using the following code we get the real world coordinates from the given (u, v) coordinates (image coordinates):

```
M = ( K*T(1:3 ,1:4))\[u; v; 1].*Z
```

We give this point as input into the FSM we designed earlier so that the arm picks and places the objects on its own without needing any input from us unlike last time when we were giving both the pick and place coordinates as input. Now the arm gets the pick coordinates from the image and we only give a place location as input.
Due to time constraints we were unable to fully execute the pick and place tasks. However, the arm was getting the correct real world coordinates and reaching to the desired location.
Point to note is that since we found the homogeneous transformation from camera to base frame by physically measuring the translation vector, it could have human errors. Also if the object is not detected properly in the image then the system could also have errors.

H. The Robot Games

We did singularity analysis and found the configurations were singularity occurs by showing that Jacobian loses rank when:

$$a_2 a_3 \sin\theta_3 [a_2 \cos\theta_2 + a_3 \cos(\theta_2 + \theta_3)] = 0$$

We found the following 3 configurations and the directions the arm cannot move instantaneously in at these configurations:

1) when**the arm is fully stretched**, it will not be able to move in any of the axes instantaneously.
2) when **the end-effector is right above the base, without the arm being stretched**, the arm will not be able to move in x and y axes instantaneously.
3) when **the end-effector is above the base and the arm is stretched** the arm is unable to move along z-axis instantaneously.

Unfortunately we were not able to perform the complex pick and place tasks correctly. However, the arm was identifying the objects and their colors but was going to the wrong pick location as there was a slight error. This could have been because of incorrect object detection from camera image or because of error due to the height of the camera that we set physically.

IV. Conclusion and Future Work

We learned about the Phantom X Pincher arm robot in this project along with Forward and Inverse Kinematics, Jacobian, and singularities. We also explored the Intel RealSense SR-305 camera and made the arm execute vision-based pick and place tasks in this project. The use of visual based sensing allowed the robot to recognise and move items in its workspace with greater accuracy. This highlights the importance of accurate robot arms in different industries.
For future work, we can make the arm perform pick nd place completely on its own by getting the place coordinates from the image itself. To minimize errors, object detection can be improved using deep learning.

V. Digital Material

GitHub repository: https:github.comnitrodragonoidArm-project-for-intro-to-robotics-lab.git

Video playlist: https:youtube.complaylist?list=PLVY5bk6-gn-rl1HEpWywplN5P0hcXMWAe

Acknowledgments

We are sincerely grateful to everyone who helped us during this project. Dr. Basit for his invaluable guidance, Ahmed Ali for providing assistance in and outside lab hours, our classmates for our collective efforts, and to the institution for access to lab facilities and equipment. We also express gratitude to the open-source community and researchers who helped and inspired us with their work.

Appendix A
Pseudo-code for getting position of cubes from image

Input: an image from the camera
Output: no. of objects of each color and their centroids

```
binarize the image we got from camera
get an image with black background and white foreground/objects
applying the connected components function → we get no. of objects and their pixel data
for i = 1:no. of objects do
    if pixel is not of a cube then
        remove it from image
separate rgb values from color image
turn image completely black
for i = 1:no. of objects do
    store mean rgb values of the objects in 3 diff. arrays
    if rgb values above a certain threshold then
        make the binary value equal to 1
    for i=1:no. of pixels do
        store non-zero pixels of all objects in an array
    plot histograms of pixel intensities
    get the max. intensity and its index from histograms data
    set threshold value of pixel intensity
    while 1 < no. of pixels of each object do
        if pixel value less than threshold then
            make the pixel value 1 (turn it white)
        else
            remove pixel from image
    get pixel value of centroid and make it black
for i = 1:no. of objects do
    if object is red then
        store in red_obj
    if object is blue then
        store in blue_obj
    if object is green then
        store in blue_obj
```

References

[1] "PhantomX Pincher Robot Arm Kit," *Trossenrobotics.com*, 2021. https:www.trossenrobotics.compPhantomX-Pincher-Robot-Arm.aspx
[2] "Depth Camera D415," *Intel® RealSenseTM Depth and Tracking Cameras*, Apr. 10, 2023. https:www.intelrealsense.comdepth-camera-d415
[3] "Pick and Place Robots: What Are They Used For and How Do They Benefit Manufacturers?," *Automate*, 2023. https:www.automate.orgblogspick-and-place-robots-what-are-they-used-for-and-how-do-they-benefit-manufacturers
[4] Basit Memon. *Introduction to Robotics Lab Handbook*, 2nd ed., 2023.
[5] "VisibleBreadcrumbs," Mathworks.com, 2023. https:ch.mathworks.comhelpimageslabel-and-measure-objects-in-a-binary-image.html