Introduction to Robotics Lab - Final Report

Hamad Abdul Razzaq, Muhammad Ali Siddiqui
Habib University, Pakistan
Email: hr06899@st.habib.edu.pk, ms06875@st.habib.edu.pk

***Abstract*—In this ARM Project, the over all picture is to be able to pick and place cubes of different colors from one place to the other. In doing so, Our Manipulator will get to know about the position of the cubes placed in the workspace with the help of an RGB and a depth camera. With the help of this information, we will apply relevant transformations, and perform calculations to determine the Joint Angles of the ARM Manipulator, and execute them to pick the cubes from their respective positions and place them at the desired location. We also take care of some bounds throughout the process such as our Manipulator doesn't come in a singularity state, the executed joints angle are practically realizable (meaning that the joint angles are executable, the manipulator doesn't collide with itself, the workspace is reachable), etc. are taken care of so that we may optimally execute our Pick and Place project.**

***Index Terms*—Phantom X Pincher, Robot Manipulator, Pick and Place**

I. Introduction

The main goal of this project is to be able to pick up the cubes of same size but different color within the reachable workspace of our ARM Manipulator, with the help of an RGB and Depth Camera and then to be able to place them at a desired position in our workspace. This simple Pick and Place task is widely used in a lot of applications. For example, in food industries, there are many ARM Manipulators whose task is to pick the food that are coming through the sliding rail and collect them by dropping them in a certain place.

Before getting started with the project, it is important to note the tools that we will be using. We will be using MATLAB for this system. The ARM Manipulator used for this project is Phantom-X Arm Pincher which has 4 revolute joints as can be seen below:

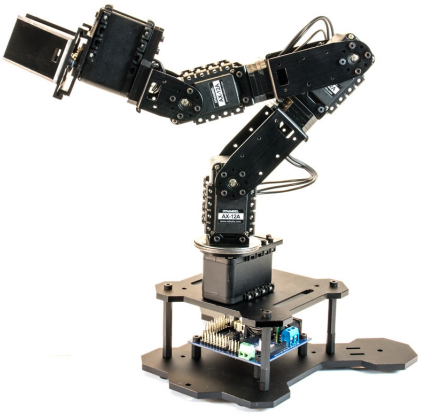

Fig. 1. Phantom-X Arm Pincher

We will also be using the Intel RealSense Depth Camera SR305 which has both an RGB and a Depth Camera. This camera can be seen below:

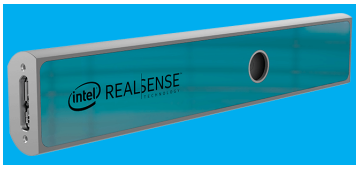

Fig. 2. Intel Real Sense Depth Camera S305

Before using the RGB image, we will scale it down to match the resolution of Depth camera.

II. Methodology

To achieve the aforementioned task, We have to first determine what is our desired output and what are the sources from which we can gather the information. In order to get a better idea of this project we divide it into the following sub-modules:

- Gathering Workspace Information
- Communicating Information to ARM Manipulator
- Commanding ARM to execute the Task

The Block Diagram of our System can be seen in the following image:

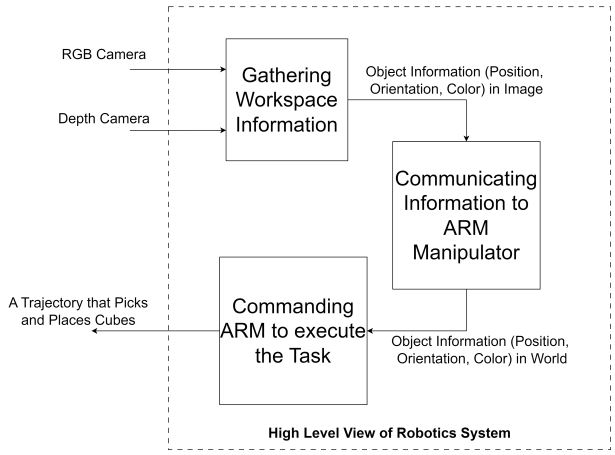

Fig. 3. Block Diagram of Robotics System

We will now look into these sub divisions of the system.

A. *Gathering Workspace Information*

The first and foremost aspect to consider is what input we have and how can we make sense of it. In this case, we used an Intel RealSense Depth Camera SR305 [1] which has an RGB and a Depth Camera. We placed the camera vertically above our manipulator pointing downwards and at such an height that the area of workspace which can be reached by our manipulator is visible.

The first task in this sub-system is to determine and extract the objects of interest in our workspace. For this project, the Objects of interest are the cubes that will be placed in random positions in our workspace. An example of an image in our workspace is shown below:

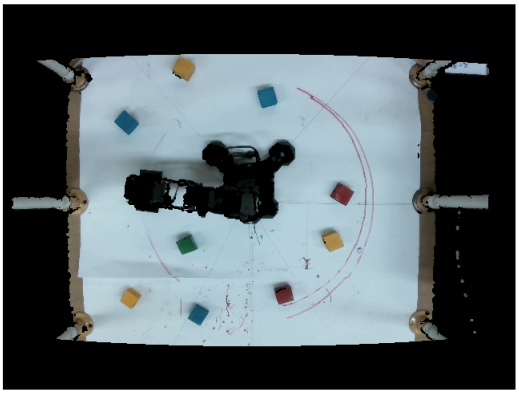

Fig. 4. Sample Image of Workspace taken from RGB Camera

*1) **Object Detection and Extraction**:* Given the workspace information as an image, our first task is to detect and extract the pixels that are constitute an object. For that we will first separate the objects in our workspace from the background with the help of first binarizing the RGB values for every pixel and then running the following pseudocode:

Algorithm 1 Separating Objects From Workspace

Input: RGB image A

Output: A Binary (Black and White) image with the same size as input image

1: Make a an empty image I
2: **for** each u, v in Image A **do**
3: $\quad I(u,v) = \neg(R(u,v,A)\,\&\,G(u,v,A)\,\&\,B(u,v,A))$
4: **end for**

Once we have the above image, we then use MATLAB's built in bwconncomp function to group the neighbouring pixels into one. This will then give us the total count fo pixels per object. From Trial and error it was known that our cube contains arounf 700 to 800 pixels on average so we set some relevant thresholds to eliminate any such object that has more or less pixels from the set threshold. An example of an image obtained after the above pre-processing is shown below:

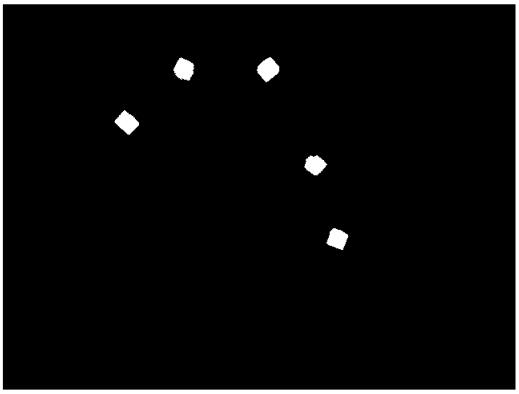

Fig. 5. Object Detection and Extraction

Now that we have our objects determined, we would now represent the object by simply a one pixel which we chose to be the centroid of the object. We did this because each cube is of same length and hence we can retrieve the remaining information by just knowing the centroid pixel value for each cube.

*2) **Top-Face Pixels Extraction**:* But before determining the centroid, we must also make sure to remove the pixels that correspond to the faces of the cube other than the top face. For that we use MATLAB's otsu threshold command to filter out the pixels whose depth value as obtained from the Depth Camera is greater than the remaining ones. This will then leave us with the pixels that make up the top face of the cube and we will simply find the average of x and y values of all pixels to find the centroid of an object. In the above example, the top face pixels of cubes and their respective centers can be seen below:

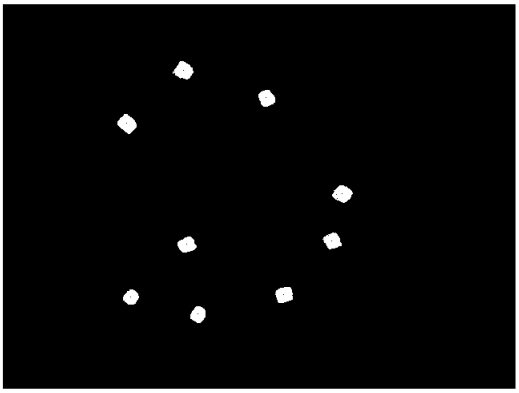

Fig. 6. Top-Face Pixel extraction and Centroid Determination

*3) **Color Determination**:* We know that in our project, we are working with a cube that has a same uniform color. In order to determine the color of an image, we set a certain threshold values for the red, green and blue channels with the help of trial and error, and then we characterize the color of each image as a binary value indicating the absence or presence of an R G or B color. In the example given above, the color determination array at the end pf the algorithm is shown below:

```
red_val_bin: [0 1 1 0 0 0 1 1 1]
green_val_bin: [1 1 1 1 1 1 0 1 0]
blue_val_bin: [1 0 0 0 1 1 0 0 0]
```

Fig. 7. Color Determination

*4) **Depth of Objects**:* Lastly, in order to determine the depth value of an object, we use the information provided by the depth camera. For each Top-face pixel of an image, we average out the depth values, giving us the final depth value of each cube in our workspace.

B. Communicating Information to ARM Manipulator

Now that we have the pixel information, color information and depth information of each object of interest in our scene, our next goal is to manipulate this information in such a saw that the robot can understand. But before that, we must be able to determine the configuration of the robot so that the given information can be transformed in a way that robot can understand. We also store the depth info in an array in the same way as shown above for the colors. But the array for depth info has to be of type double so that it can store floating point values.

*1) **Configuration of Robot**:* We determine the configuration of the robot by Assigning DH frames and determining DH parameters to the joints. As a result of this assignment, we will have a base frame and a set of Transformations A_i which will help us to transform from one joint frame to the other. This will eventually lead us to the last frame determining the position of the grasper of ARM Manipulator. The following are the assigned frames and the respective parametes for joint angles.

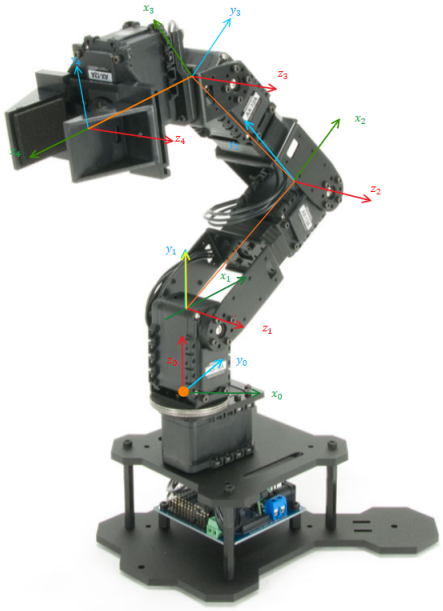

Fig. 8. DH Frame Assignment

Link No.	a_i (mm)	α_i (radians)	d_i (mm)	θ_i (radians)
1	0	$\frac{\pi}{2}$	54	θ_1
2	108	0	0	θ_2
3	108	0	0	θ_3
4	76	0	0	θ_4

Fig. 9. DH parameters for out Manipulator

*2) **Transforming Image Information into Robot's Configuration**:* As mentioned earlier, there will be a base frame assigned to the robot. We have to find out the position of the object with respect to this frame so that the ARM can make sense of it. In order to do so, we get the Intrinsic parameters of the camera to build the Camera Calibration Matrix K and also find the transformation from the Camera frame to the base frame by physically measuring offsets. Once we have both these information, we can transform our pixel information into 3D world coordinates using the following equation:

$$\begin{bmatrix} Zu \\ Zv \\ Z \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

This will give us the position of object in the base frame. From this position, we have to set the four joint angles of our ARM Pincher $\theta_1, \theta_2, \theta_3, \theta_4$ in such a way that the grasper reaches the cube's position.

C. Commanding ARM to execute the task

Our ARM Picher have 4 joint angles and it only understands the values of Joint angles, meaning that it works in Joint Space,

not in Work space. Therefore, we have to now transform the workspace information i.e., the position of the cubes into Joint Space to execute the Pick and Place Operation.

*1) **Applying Inverse Kinematics to reach the Object**:* We use the underlying geometric approach to determine the joint angles $\theta_1, \theta_2, \theta_3, \theta_4$ given the end-effector position and orientation (x, y, x, ϕ). The position here is the centroid of cube and the orientation should by such that the grasper is able to grasp the object and pick it up which is also know to us before hand.
Using the above information, we apply the Inverse Kinematics and get 4 possible sets of Joint angles that can achieve the desired position and orientation. Now the choice is ours to choose the most reasonable set. For this project, we take into account the current values of joint angles and choose the solution which will cause lesser change in the Joint angles. This will give us the optimal solution which we will pass on to the ARM Manipulator

*2) **Designing FSM to Pick and Place the Object**:* Now that we know how to reach the object via Robot ARM, we look forward to design a Finite State Machine that does this job for us. Following are the states of our FSM:

- State 1: ARM is in Rest position (We make sure this is not a singular position)
- State 2: ARM receives Pick position, performs inverse kinematics and finds the optimal solution. The ARM then moves to the position of the centroid of the cube, but just some δ units above the z value of the object and opens the grasper.
- State 3: Our ARM is set to pick the object up, we decrease the z value by δ. This will make our grasper go around the cube. We then proceed to close the grasper so that the Object is acquired.
- State 4: ARM receives the Place Position, this can me any predefined position in the workspace. The ARM performs the same step, i.e., to apply inverse kinematics, then to find the optimal solution and move at a certain offset δ above the actual Place position.
- State 5: ARM moves down by δ units and opens the grasper to release the object. Before going to IDLE State, ARM first goes up by δ units to make sure the object is not displaced from its place position. Then finally, the ARM returns to its IDLE Position

The States in above FSM was broken down into smaller states for better modularity of the code. This state Diagram of this FSM can be seen below:

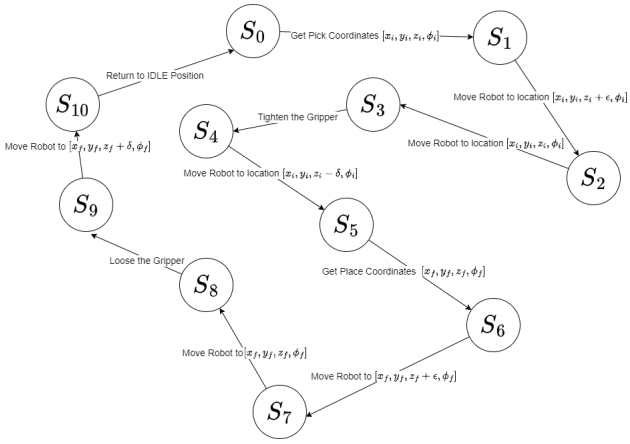

Fig. 10. FSM for Pick and place Operation

*3) **Picking and Placing Objects based on Color**:* Now that we have a system built for picking and placing cubes in our workspace. The next step is to pick up the cubes of certain color and place them in a certain Place position and so on. As we already have the color information for all the objects, we can simply utilize that information to characterize the cubes based on their colors and thus, place them at the locations in accordance with the color that they have.

III. Results

In order to determine results, we must know that a successful testis the one in which we are able to pick up the cube of a certain color and place them at the desired position. We can break this test into smaller sub-tests as follows:

A. Color Detection Test

This test evaluates the performance of our system regarding the correct color detection of the code. It was observed that for about 80 tests conducted, our system was able to detect the correct color for 75 tests.

The reason of failure for the remaining tests depends on the lighting conditions, shadows, etc. that alter the color information of an object in the RGB image. In our case, the failed test was because of the shadows that made the color of our object dull to an extent that it was not able to differentiate between black and colored objects. This problem arose the most for the red color, since the red color used is a dark red color and resembles the black color to an extent.

The possible fixes could be to have a uniform lighting around the workspace and avoid unnecessary objects in the surroundings that cause the shadows to occur. Furthermore, it is also to notice that our System will fail to detect color if the intensity of light is changed by a great amount (increased / decreased).

B. Pick Operation Test

The success of this test means that the object in a workspace is picked up successfully by the ARM manipulator within its

reachable workspace. The reachable work space of ARM is Spherical as shown below.

Upon testing the Pick operation for multiple times, it was observed that about 5 out of 35 times our ARM succeeded in picking up the object. However, it was noted that the ARM Manipulator did got nearer to the pick position but not exactly at the pick position. There could be many reasons for this error among which the primary are:

- Incorrect Centroid Determination of Cube
- Orientation of Cube in such a way that the Grasper is unable to grasp it
- Inaccurate Transformation from the Camera frame to the base frame.
- Human Error in measurement. This is the most reasonable error given the fact that the ARM manipulator is reaching closer to the final position but not exactly at the final position
- Mounting of Camera. Since the camera was mounted in a way that it can be rotated about its x axis, therefore unintentional rotation of camera by slight degrees can cause a significant amount of error in determining the position of object in world frame.

Quantitively, it was determiend on average that for a position (x, y, z) of an object, our ARM Manipulator reached $(x, y, z) \pm 10$mm for about 90% of the tests (32 our of 35)

C. *Place Operation Test*

The Place Operation test is a success if the already grasped object in placed at the given position or not. This test has been successful for almost all of the time (36 out of 40) given that the place position is practically realizable and reachable. However, as discussed in the previous section, there were some errors offsets of ± 10mm along all the three axes.

D. *Overall Performance Analysis*

From the sub-tests given above, it can be implied that our Robot will not be able to execute the Pick and Place operation correctly for most of the time. Although we cannot reach the desired result, but we do see that our ARM does all of his jobs correctly but with some significant error which disables it to execute Pick and Place in a correct way. If this errors were removed, it can be calculated and seen that for about $90 + \%$, our System can perform the desired task. The remaining percent incorporates the uncertainties in measurement, the lighting conditions, etc.

IV. Conclusion and Future Work

In Short, Our system performs a basic Pick and place operation with the help of an ARM Manipulator. In doing so, it first extracts the object and their position and color information with the help of an RGB and Depth Camera. It then moves on to transform this information into 3D world coordinate system upon which we can apply inverse kinematics to transform convert position into joint angles which our robot can make sense of. Lastly, we execute the optimal set of joint angles in such a way that the ARM moves towards the object, picks it with its grasper and places the object at its place position. The future work could be to generalize this for objects of any shape and size as long as they can be grasped by the grasper. One could also look into picking and placing dynamic objects, i.e., the objects are not static anymore and that they are moving as we are detecting them. Hence, we get to see that this basic pick and place operation has a lot of significance and can be extended to incorporate complex scenarios as well.

V. Digital Material

The link for all the sources, demos and guides can be find below:
https://github.com/Hamad-Abdul-Razzaq/Robotics_Lab

Acknowledgments

I would like to thank the following Individuals for helping me in this project:

- Dr. Abdul Basit Memon
- Sir Ahmed Ali Mustansir
- Muhammad Ali Siddiqui
- Syeda Manahil Wasti
- Khuzaima Ali Kham
- Syed Jahania Shah
- Muhammad Suleiman Qureshi
- Syed Muhammad Mustafa

References

[1] Intel RealSense Depth Camera SR300 Series Product Family Datasheet
[2] Manufacturer provided arm resources: https://learn.trossenrobotics.com/
[3] Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. John Wiley Sons, 2020.
[4] https://www.mathworks.com/help/images/label-and-measure-objects-in-a-binary-html