Robotic Pick and Place System Using a 3-DOF Manipulator

Samara Holmes, Haotian Liu, and Lehong Wang
Robotics Engineering Department (RBE)
Worcester Polytechnic Institute
{sdholmes, hliu8, lwang11} @wpi.edu

***Abstract*— The objective of this project is to create a system to identify targets of assorted colors, localize them with respect to the robot, and grab and sort them. To complete this objective, there needs to be an implementation of camera calibration of a vision system, identification of targets in the coordinates of the robot, and organization of targets all within the MATLAB platform.**

I. Introduction

This robotic pick and place system will require the design of system architecture and workflow in MATLAB. Using camera vision will allow for the identification of objects on a checkerboard for the robot to pick up and sort based on color. The robotic manipulator will require experimentation and MATLAB functionality to identify and localize the targets efficiently and autonomously.

A. Motivation

The motivation to pursue this project comes from the experimentation with the aspects of a robotic pick-and-place manipulator on a smaller scale. Not only will this approach provide hands-on experience with a robotic manipulator, but it will also focus much of the efforts on camera vision and how to identify/manipulate objects within a robot's workspace.

B. Background

Calculating the forward velocity kinematics, inverse velocity kinematics, and DH parameters will be required prior to completing the final demonstration.

Denavit-Hartenberg (DH) parameters are used to mathematically describe the coordinate frames for the links of a robotic system. To obtain the DH parameters, Fig. 1 shows the drawing used to identify link lengths, reference frames, joint variables, and transformation matrices.

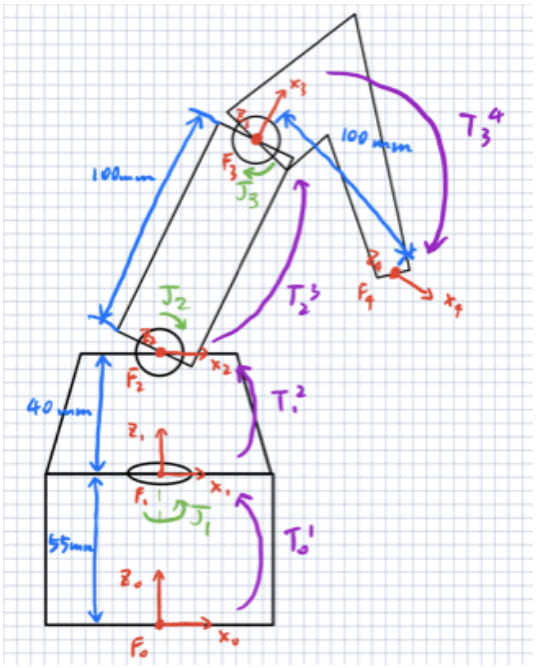

Fig. 1. Initial drawing of the manipulator.

Table I shows the DH parameters determined for the 3-DOF robotic manipulator based on Fig. 1.

TABLE I. DH Parameter Table

	θ	**d (mm)**	**a (mm)**	**α**
T0 to T1	0	55	0	0
T1 to T2	θ_1	40	0	-90
T2 to T3	θ_2-90	0	100	0
T3 to T4	θ_3+90	0	100	0

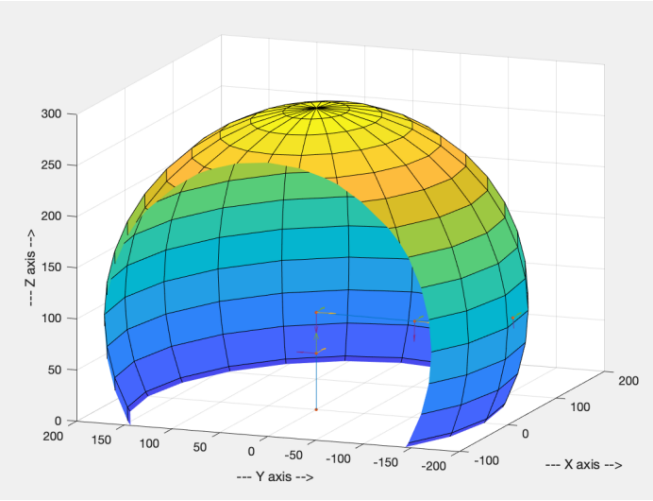

Fig. 2. Workspace of the manipulator.

Fig. 2 shows the approximate workspace of the robotic manipulator.

Fig. 3 clearly illustrates the derivation of the inverse kinematics of the arm. Not only did the derivation need to utilize 'atan2()' to calculate the joint angles, but the code also needed a 'check_in_work_space()' function that ensures that the position values given are within the robot's workspace.

```
function I = ik3001(pos)

    a1 = 100;
    a2 = 100;
    d1 = 95;
    x = pos(1);
    y = pos(2);
    z = pos(3);
    r = sqrt((x^2) + (y^2));
    s = z - d1;

    Matrix.check_in_work_space(x,y,z);
      in_workspace = Matrix.check_in_work_space(x,y,z);
      if  ~in_workspace
          return
      end

    D1 = x/r;
    C1 = sqrt(1 - (D1^2));
    theta1 = atan2(y,x);

    alpha = atan2(s,r);
    D2 = ((a1^2) + (r^2) + (s^2)- (a2^2)) / (2*a1*(sqrt((r^2) + (s^2))));

    C2 = -(sqrt(1 - (D2^2)));
    beta = atan2(C2,D2);
    theta2 = pi/2 - (alpha - beta);

    D3 = -((a1^2) + (a2^2) - ((r^2) + (s^2)))/(2*a1*a2);
    C3 = sqrt(1 - (D3^2));
    theta3 =  (atan2(C3,D3))- pi/2;

    if theta2 <= -pi/2 || theta2 >= pi/2
        theta2 = pi/2 - (alpha - (atan2(-C2, D2))); % safety checks for angles 2 and 3
    end
    if theta3 <= -pi/2 || theta3 >= pi/2
        theta3 = atan2(-C3, D3) - pi/2;
    end

    I = [theta1, theta2, theta3];
    I = I * (360/(2*pi));

end
```

Fig. 3. 'ik3001()' code.

Below are the calculations for the 6 x 3 Jacobian matrix. The Jacobian uses the joint velocities of the robot to return the end-effector velocities. The upper half of the Jacobian represents the linear velocity of the end-effector, while the lower half represents the angular velocities of the end-effector. To shorten the equations, the variables 's' and 'c' are used for 'sine' and 'cosine' respectively.

$$J_{11}: 100s(\theta_1)s(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_2 - \tfrac{\pi}{2})c(\theta_3 + \tfrac{\pi}{2})s(\theta_1) - 100c(\theta_2 - \tfrac{\pi}{2})s(\theta_1)$$
$$J_{21}: 100c(\theta_1)c(\theta_2 - \tfrac{\pi}{2}) + 100c(\theta_1)c(\theta_2 - \tfrac{\pi}{2})c(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_1)s(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2})$$
$$J_{31} = 0$$
$$J_{12}: -100c(\theta_1)s(\theta_2 - \tfrac{\pi}{2}) - 100c(\theta_1)c(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_1)c(\theta_3 + \tfrac{\pi}{2})s(\theta_2 - \tfrac{\pi}{2})$$
$$J_{22}: -100s(\theta_1)s(\theta_2 - \tfrac{\pi}{2}) - 100c(\theta_2 - \tfrac{\pi}{2})s(\theta_1)s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_3 + \tfrac{\pi}{2})s(\theta_1)s(\theta_2 - \tfrac{\pi}{2})$$
$$J_{32}: 100s(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_2 - \tfrac{\pi}{2})c(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_2 - \tfrac{\pi}{2})$$
$$J_{13}: -100c(\theta_1)s(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_1)c(\theta_3 + \tfrac{\pi}{2})s(\theta_2 - \tfrac{\pi}{2})$$
$$J_{23}: -100c(\theta_2 - \tfrac{\pi}{2})s(\theta_1)s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_3 + \tfrac{\pi}{2})s(\theta_1)s(\theta_2 - \tfrac{\pi}{2})$$
$$J_{33}: 100s(\theta_2 - \tfrac{\pi}{2})s(\theta_3 + \tfrac{\pi}{2}) - 100c(\theta_2 - \tfrac{\pi}{2})c(\theta_3 + \tfrac{\pi}{2})$$

$$J_{41}: 0$$
$$J_{51}: 0$$
$$J_{61}: 1$$
$$J_{42}: -s(\theta_1)$$
$$J_{52}: c(\theta_1)$$
$$J_{62}: 0$$
$$J_{43}: -s(\theta_1)$$
$$J_{53}: c(\theta_1)$$
$$J_{63}: 0$$

II. Methodology

A. Camera Setup and Intrinsic Calibration

The camera is set up using the MATLAB command, 'webcamlist', to return a list of digital imaging sensors "visible" to MATLAB. Then the command 'cam = webcam(2)' is used to generate a camera object, and the command, 'preview(cam)', is used to display the camera feed.

Intrinsic calibration obtains the intrinsic parameters in the camera matrix. These parameters, including focal length pixel dimension, etc. will enable the ability to use the camera as a sensor to take real-world measurements. Using the following instructions, the camera was calibrated:

1. Open the MATLAB camera calibrator.
2. Take at least 60 pictures while moving the camera around the checkerboard in several different orientations.
3. Go to the 'calibration' tab and use approximately 20 photos with properly aligned axes
4. Select the 'fisheye' camera model and press 'calibrate'.
5. Evaluate the results.

B. Extrinsic Calibration

Extrinsic calibration is the performance of registration between the robot's reference frame and the image's reference frame (see Table II for frame descriptions). This will relate the position of the targets within the field of view of the camera to robot task coordinates. Using the following instructions will extrinsically calibrate the camera/robot:

1. Run the 'getCameraPose()' function to determine the extrinsic calibration parameters of the camera.

2. Obtain a position of interest in x-y pixels in the image frame using the data tips tool.
3. Use 'pointsToWorld()' to transform the image frame into the checkerboard frame.
4. Obtain the transformation matrix from the robot's base frame to the checkerboard's frame.
5. Multiply the transformation matrix by the output of the previously used function 'pointsToWorld()'.
6. Perform validation on the camera-robot calibration registration.

TABLE II. FRAME NAMES AND DESCRIPTIONS

FRAME NAME	DESCRIPTION
F_0	Robot's base reference frame
F_{Image}	the reference frame of the image plane (pixel coordinate)
$F_{Checker}$	Local reference frame of the checkerboard

C. Object Detection and Classification

Using image processing methods, objects will be identified after determining an image processing pipeline to take an image from the camera, separate the targets from the board, identify the target's centroid, and overlay a marker on the camera feed. This pipeline of functions will return the estimation of the coordinates of the objects with the requested color.

D. Object Localization

The targets being used can now be localized by converting the centroid location to usable positions for the robot arm to move to. In this step, we take the estimated coordinates from the previous step and analyze and reverse multiple primary causes of the error to make the estimation closer to reality.

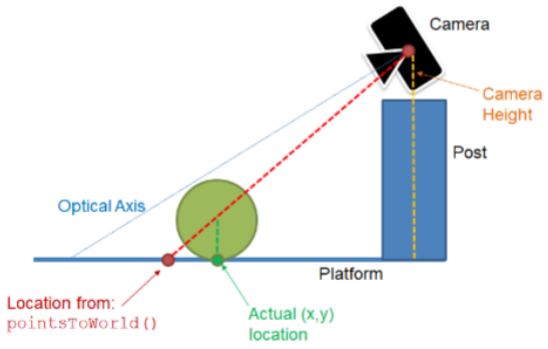

Fig. 4. Object localization diagram.

E. Final Demonstration

The final demonstration consists of implementing a script in MATLAB that:

1. Determines the 3D position of an object with respect to the robot's reference frame.
2. Uses inverse kinematics to calculate the joint angles required for the end-effector to reach the targets.
3. Uses trajectory planning to implement a smooth motion from the robot's position to the targets.
4. Closes and opens the gripper properly to pick and place the targets.

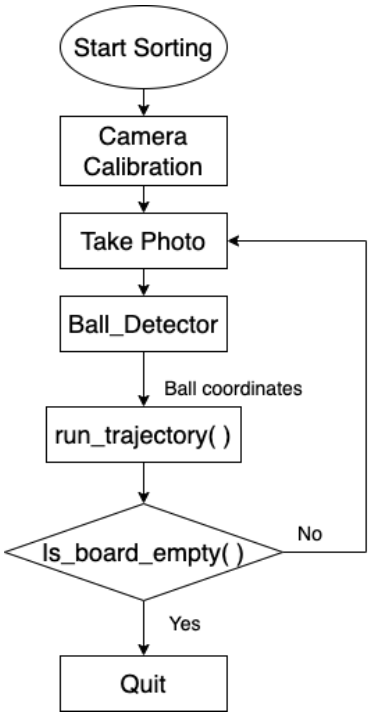

Fig. 5. Sorting workflow diagram.

The final demonstration requires a system to sort objects based on color. The manipulator should pick up all objects in the field of view and move them to their designated locations in a smooth manner (using polynomial trajectories, inverse velocity kinematics, etc). The sorted balls may still be in the field of view of the camera, so the code should account for that corner case.

Shown in Fig. 5 is the final demonstration workflow. When the function is called to begin sorting, the robot will take a photo of the workspace, use the Ball_Detector class to return the target's coordinates, then feed those coordinates into the run_trajectory() function to perform cubic trajectory planning, and then check if the board is empty. If the board is empty, it will quit the process, if there are still targets in the workspace, the robot will repeat the sorting process.

III. RESULTS

A. Intrinsic Calibration

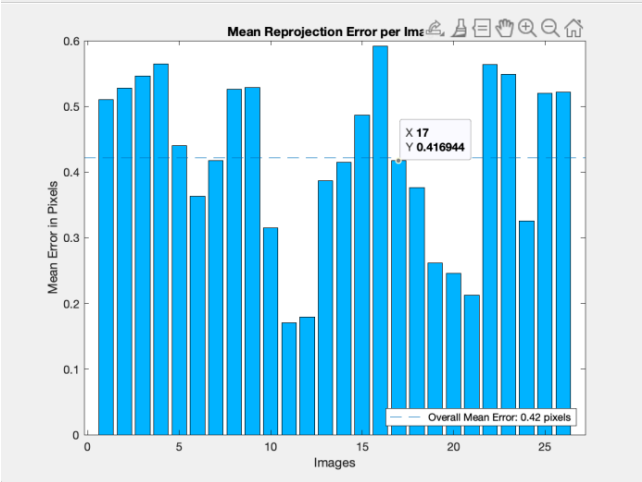

Fig. 6. Mean reprojection error for 26 images.

Fig. 5 shows the mean reprojection error for the 26 images used in the calibration of the camera. This reprojection error represents the distance between a calibration image point and a world point projected onto the same image. Upon seeing an error greater than 1, the calibration image outlier would be removed from the dataset and recalibrated.

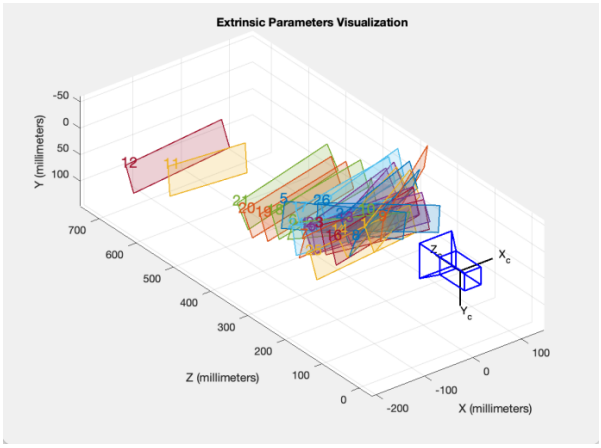

Fig. 7. Checkerboard-centric extrinsic parameter visualization.

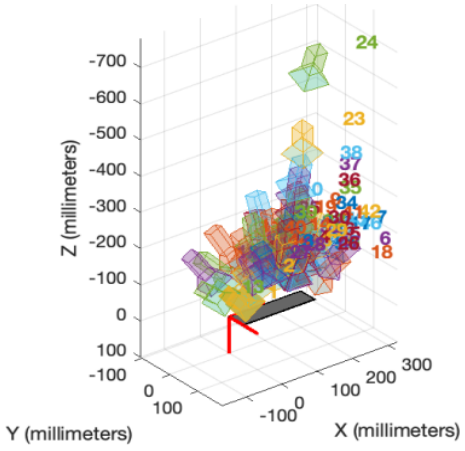

Fig. 8. Camera-centric extrinsic parameter visualization.

Fig. 7 shows a MATLAB plot visualization of 26 calibration images. It is checkerboard-centric, meaning it shows the angle of the checkerboard for each image, keeping the camera static.

Fig. 8 also shows a MATLAB plot visualization of the 26 calibration images, however, it is camera-centric, meaning it shows the camera angles at which the images were taken, keeping the checkerboard static.

B. Extrinsic Calibration

Using the 'getCameraPose()' method, the transformation from the image plane to the checkerboard frame could be calculated. Using this method along with the transformation from the robot's base frame to the checkerboard's frame, the transformation from the image frame (x-y) to the robot's reference frame (x-y-z) can be obtained.

C. Object Detection and Classification

Fig. 9. Objects placed on checkerboard intersection points.

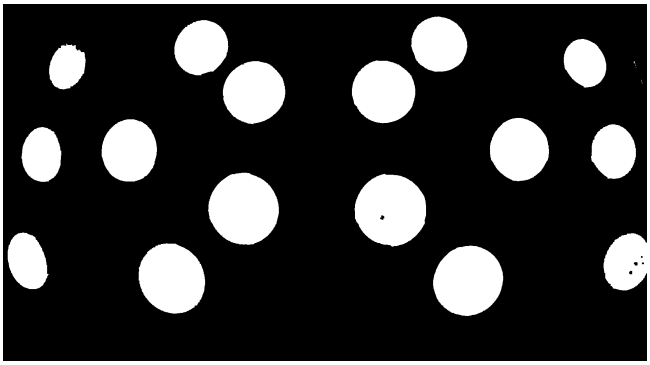

Fig. 10. Binary image.

Fig. 11. Colored image mask.

Fig. 9 shows the targets in assorted colors placed on the checkerboard intersection points. Since the camera uses a fisheye lens, the image needs to be undistorted in MATLAB.

Fig. 10 shows the binary image mask of the targets generated by applying HSV color space filters to find all the colored balls.

Fig. 11 shows the mask shown in Fig. 10 being applied to the original image, reviewing only the colored targets.

D. Object Localization

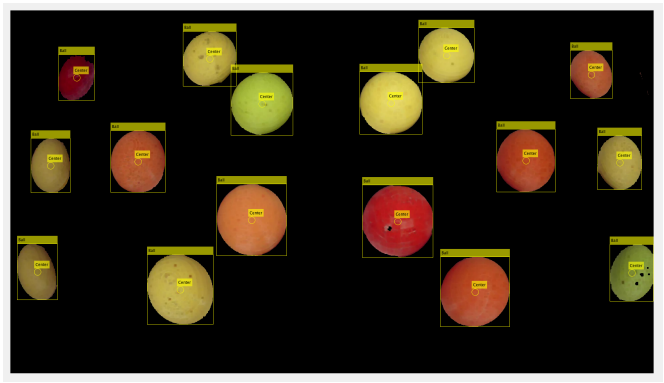

Fig. 12. Finding the centroid.

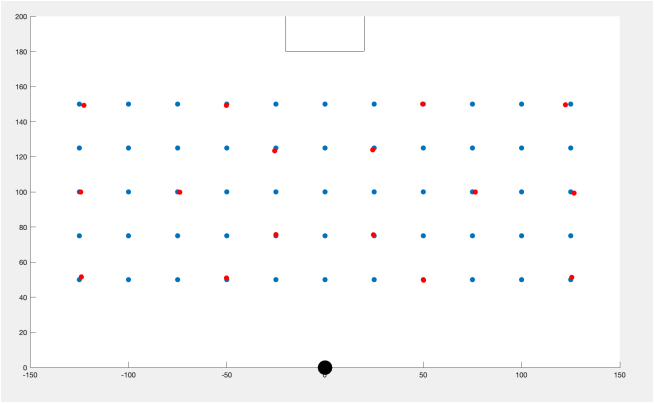

Fig. 13. The centroid of the targets vs the reprojected centroids (camera error).

Fig. 12 shows the centroid and boundary being drawn on the objects. This data will be used to convert the centroid to usable target positions for the robot to pick up.

Fig. 13 represents the error between the reprojected centroids of the balls and their real coordinates. This graph works because the balls are placed exactly over the intersection points of the checkerboard.

E. Final Demonstration

The final demonstration challenge was to use the 3-DOF manipulator to autonomously sort objects of assorted colors. This was completed using a combination of forward and inverse kinematics, along with trajectory planning and image processing.

Errors occurred when the workspace wasn't properly lit, however, it did help that the targets were bright colors and very different from the colors of the checkerboard. For example, a black target would not be detected on the checkerboard because it is the same color as the workspace.

Another significant error needed to be accounted for is that since the camera has a fisheye lens, the targets on the edges are significantly distorted, and their actual centers are different than what is shown in the image. This effect can be observed in Fig. 13, where targets on the edge have more errors than targets close to the center. We could have accounted for that by applying another layer of lens distortion by measuring a bunch of balls and deriving a transformation matrix from the errors. However, while testing, the arm was able to firmly and consistently grab the targets despite the error. We found this error to be negligible.

The targets were also flat on the bottom (so they wouldn't roll around) but, this means that if the ball was picked up far away and placed close to the robot's origin, the ball would sometimes roll around (since it is placed at an inadequate angle).

IV. Discussion

A. Intrinsic Calibration

It was determined that MATLAB uses the information from the pictures of the checkerboard along with the dimensions of the checkerboard to calculate the distortion parameters of the camera. The reprojection error that

resulted from the images got better as the number of images increased, however, due to the law of diminishing returns, it was better to limit the number of images to approximately 20 for the best results. As previously stated, the reprojection error histogram shown in Fig. 6 represents the distance between a calibration image point and a world point projected onto the same image.

B. *Extrinsic Calibration*

When calibrating the robot through extrinsic calibration, it was determined that the transformation from the image frame to the robot's frame would only work well when z = origin of the checkerboard reference frame's z. Since the image is 2D, it can't give useful information about the z-axis. If the z-axis changes, it will influence the other axes and return the wrong information.

C. *Object Detection and Classification*

The image processing method used consisted of HSV color space filtration, blob detection, and color thresholding. HSV color space filtration refers to the hue, saturation, and value of an object. Using this filtration method allows us to search for these values instead of RGB values. Blob detection results in a binary image (Fig. 11), and identifies the important regions. Fig. 10 is the result of masking the binary image back onto the original image. From there, the objects can be localized.

D. *Object Localization*

After detecting the objects, localizing them was as simple as finding the centroids and drawing boundary boxes around them. From there, it was necessary to use the target centroids and relate them to the checkerboard plane.

An important factor in localizing the targets is compensating for their height and the result of projecting that height to the surface of the board. If this calculation had not been made, the camera would interpret the center of the ball as if it were flat, resulting in a significant error.

E. *Final Demonstration*

In the final demonstration, the robot performed a cubic trajectory every time it sorted. The cubic trajectory is used because the start and end velocities of the end-effector could be controlled. A quadratic trajectory is not needed because acceleration is not a significant factor in our use case.

To exclude the targets that were already sorted, the sorted targets were placed outside the checkerboard, and the x-y coordinates were checked against the coordinates of the board to ensure the sorted targets were not within the workspace.

As an add-on to the project, the manipulator was programmed to not only complete the basic color sorting challenge, but also was enabled to perform dynamic object tracking and tic-tac-toe.

Dynamic object tracking involved fast calculations of the location of the target and required the robot to move the end-effector to the target's position in under a second. As for tic-tac-toe, the robot needed to take into account the targets placed on the board by a human, and the optimal position to place an opposing target.

V. Conclusion

Several methods and topics were explored in the completion of this project, including inverse kinematics, forward kinematics, trajectory planning, reference frame transformations, and image processing. Though this project mainly focused efforts on coding in MATLAB, the use of the built-in apps was an integral part of completing this project successfully.

Furthermore, this type of manipulator can be programmed to do so many other tasks including games (like tic-tac-toe), or more advanced systems such as automated assembly.

Appendix

TABLE III. Authorship Table

Task	**Contributor(s)**
Camera setup and intrinsic calibration	Lehong
Camera-robot registration (extrinsic calibration)	Haotian, Lehong
Object Detection and Classification	Haotian, Lehong, Samara
Object Localization	Haotian, Lehong, Samara
Final Demonstration	Haotian, Lehong, Samara
Abstract	Samara
Introduction	Samara
Methodology	Haotian, Lehong, Samara
Results	Haotian, Lehong, Samara
Discussion	Haotian, Lehong, Samara
Conclusion	Haotian, Lehong, Samara
Final Project Video Demonstration	Samara

TABLE IV. Submission Links

Description	**Links**
Final Project video	https://youtu.be/dcFBB7VuyJo
Github Repository	https://github.com/RBE3000X-Lab/RBE3001_Team06