University of North Carolina Asheville
Journal of Undergraduate Research
Asheville, North Carolina
May 2023

Application of Robotic Arm Control using Computer Vision and Object Detection Techniques

Sarah Vyvyan
Engineering - Mechatronics
The University of North Carolina Asheville
One University Heights
Asheville, North Carolina 28804 USA

Faculty Advisor: Dr. Eli Buckner

Abstract

The increased technological advancements in and utilization of robotic systems have greatly improved efficiency and safety in many areas of society, from industrial and manufacturing environments to medicine and healthcare. These robotic systems are often used to recognize or search for certain objects and perform actions on them, such as finding an object on a conveyor belt and moving it to a new location. Embedded cameras and computer vision systems are often implemented in this object detection and recognition process. This project demonstrates this concept by having a 4-jointed robotic arm with an embedded RGB D camera to recognize an object, specifically a human hand in front of a white backdrop, and determine its 3D location in space. The process of determining the object's location implements a simple threshold-based image segmentation method as well as uses the depth image provided by the embedded camera. Once a 3D location of the hand is determined, the arm will move the end effector to that location and follow the hand. This is done by controlling the angles of rotation of the arm's 4 joints through numerical inverse kinematics. The image segmentation method was tested on 113 different images of hands with solid white backgrounds. This method achieved a 98.2% accuracy in detecting a hand and a 95.5% accuracy in locating the center of mass within the actual boundaries of the hand. The inverse kinematics

successfully moved the end effector of the robot within range of grabbing the hand 100% of the time and successfully moved it within 5 centimeters of the hand 70% of the time.

1. Introduction

Robotic systems have been implemented in many different environments where they must have the ability to detect and respond to their surroundings. This ability is beneficial in many different assistive roles, from assisting factory workers by being able to detect and move large objects on a conveyor belt to being able to help someone who has lost a limb or suffered from other neuromuscular trauma by having prosthetics and exoskeletons that can predict human movements based on the surrounding environment. This ability or task can be accomplished through computer vision and object detection, implementing methods as complicated as neural networks to more simple methods such as color threshold-based segmentation. Implementation of and advancing these different object detection methods within robotics is a topic of great interest and continuing research. This report explores the use of image segmentation, the process of taking an image and segmenting it into different sections, for example, the foreground and background [1], to determine the location of an object and move the end effector of a robotic arm to that location. First, this report will use a threshold-based-image segmentation method that will be used to determine the 2D center of mass of someone's hand. The image or video of the hand will be captured using an embedded RGB D (red, green, blue, and depth) camera that is located on the end of the robotic arm. Once the center of mass of the hand has been located, the end effector of the robotic arm will be moved to its location. This will be done by implementing a numerical inverse kinematics method that will predict the needed positions and amounts of rotation for the robotic arm’s 4 joints that will minimize the distance or error between the location of the hand and the end effector, and bring the robot to the hand. Finally, this report will examine the results and accuracy of the use of this image segmentation method in object detection and the success of the inverse kinematics in moving the robotic arm to the necessary location.

2. Materials and Specifications

2.1. Hardware

2.1.1 Quanser QArm

This project will be using a Quanser QArm as the robotic arm to be controlled. The QArm is a 4-degree-of-freedom robotic arm with an embedded RGB D (red, green, blue, and depth) camera and a tendon-based two-stage gripper on the end. The QArm can be interfaced through MATLAB® and Simulink® as well as through Python™ and ROS™. This project will be using MATLAB® and Simulink® to interface with and control the arm. The

QArm can be controlled by adjusting the position of its joints (position mode) as well as by adjusting the current (current mode) [2]. In this project, the position mode will be the main interface for control. When in the zeroed position, the end effector sits at a height of 0.49 meters and at a distance of 0.365 meters horizontally from the base. The joint rotation limits are listed in the table below.

Figure 1. Labeled image of the Quanser QArm

Table 1. Joint Limitations of the Quanser QArm

Joint Limitations (radians)	
Base	+/- 2.967 rad
Shoulder	+/- 1.843 rad
Elbow	- 0.658 rad / +1.309 rad
Wrist	+/- 2.793 rad

2.1.2 RGB D camera

To see and detect the robotic arm's surroundings this project will be using the embedded RGB D camera located on the QArm. This embedded camera is the Intel® RealSense™ Depth Camera D415. This camera will provide both an RGB image and a depth image. The depth output resolution has a maximum of 1280 by 720. The RGB frame resolution is 1920 by 1080 [3]. The camera has a depth operating range of around 0.16 meters to 10 meters [4].

Figure 2. Image of the Intel® RealSense™ Depth Camera D415

2.2.Software

2.2.1 MATLAB® and Simulink®

To interact with and control the robotic arm, this project will implement the use of MathWorks® MATLAB® and Simulink®. MATLAB® is a programming and numeric computing platform that is often used in control systems and computer vision applications [5]. Simulink® is a MATLAB®-based graphical user interface from MathWorks® that uses block diagrams for a model-based design for systems [6]. Simulink® is used often for simulations. The Quanser QArm is compatible with both MATLAB® and Simulink® making this programming language and interface a reliable and relatively easy-to-integrate choice for the project.

3. Conceptual Framework

3.1. Image Segmentation

One very essential component to successfully having the robotic arm locate a hand in its surroundings is object detection. This project implements a simple image segmentation method to achieve successfully locating a hand. This image segmentation method is based on a threshold. The camera on the robotic arm collects RGB D images. An image containing red, green, and blue values is collected as well as an image containing depth values in meters is collected.

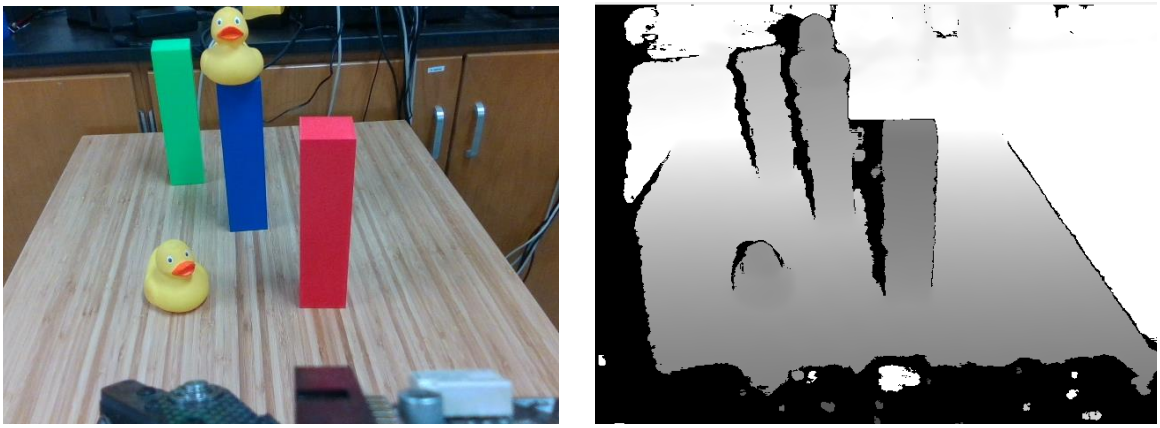


Figure 3. Example of RGB versus depth image from the camera. The image of the left is the RGB image of several objects sitting on the top of a table. The image on the left is the depth image of the same scene.

The initial approach was to compare the red and green values for each pixel in an image and find a common ratio between the two values that is true for a majority of skin tones. After examining 150 different images of hands, it was determined that a consistent ratio between these two values was 1.2. This means that when there is an object in an image that is a color similar to one's skin it has a red value that is 1.2 times the green value. Therefore, when analyzing the image provided by the camera, this method will search for pixels containing a red value that is 1.2 times the green value and label that pixel as being part of a hand. All other pixels will be considered the background and ignored in further analysis. A matrix the size of the image is created containing a value of 1 if the pixel is part of a hand and a 0 if it is not.

Figure 4. RGB image of hand compared to the segmented image. The image on the left is the RGB image of the hand. The image on the right is the black and white segmented image of the same hand. Everywhere in the image where a hand is detected is white, the rest is black.

Once it has been determined that a pixel is part of a hand or not, that information will be used to determine if a hand is present and then used to calculate the 2D center of mass of the hand.

3.2. Inverse Kinematics

This project implements the use of forward and inverse kinematics to move the end effector of the robotic arm to the location of the hand. Inverse kinematics uses kinematic equations to determine the motion required for a robot to reach a desired position [7]. The first step in inverse kinematics is to set up the forward kinematics, which is done by defining coordinate frames for each joint using the DH (Denavit-Hartenberg) convention and determining the parameters needed to calculate the poses, or the position and orientation, of each joint [1, p. 196]. Below is a model of the robotic arm (Figure 5) with the DH coordinate frames and parameters defined. The parameters defined in table 2 are used in equation (1) to calculate the pose of each joint with respect to the previous joint [1, p. 198-199].

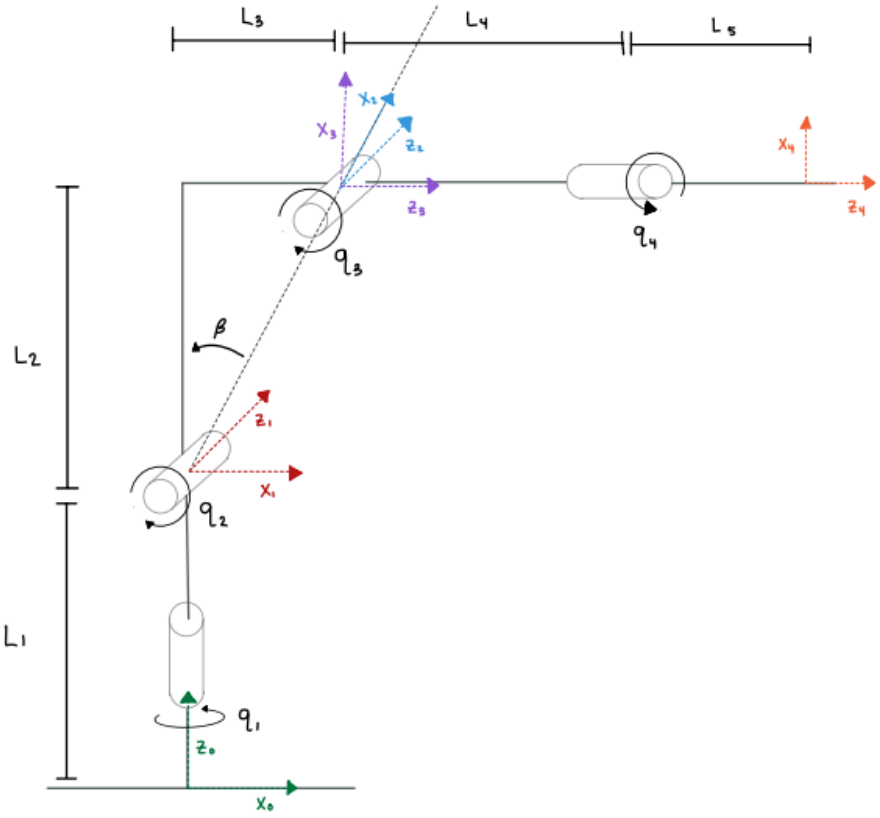


Figure 5. Simple Model of QArm with assigned DH frames for each joint including variables representing actuation parameters (rotations of each joint) and variables representing the length of the different sections of the arm.

Table 2. The DH parameters used to generate the poses of each joint. Each joint has two rotational parameters and two translational parameters. These are based on the rotations of each joint and their relative distances and angles to one another.

Joint	θ_j	d_j	r_j	α_j
1	q_1	L_1	0	$\frac{-\pi}{2}$
2	$q_2 + \beta - \frac{\pi}{2}$	0	$\sqrt{L_2^2 + L_3^2}$	0
3	$q_3 - \beta$	0	0	$\frac{-\pi}{2}$
4	q_4	0	0	0
End Effector	0	$L_5 + L_4$	0	0

(1)

$$^{j-1}\xi_j = R_z(\theta_j) \oplus \mathrm{T_z}(\mathrm{d_j}) \oplus \mathrm{T_x}(\mathrm{r_j}) \oplus \mathrm{R_x}(\alpha_j) = \begin{bmatrix} \cos(\theta_j) & -\sin(\theta_j)\cos(\alpha_j) & \sin(\theta_j)\sin(\alpha_j) & r_j\cos(\theta_j) \\ \sin(\theta_j) & \cos(\theta_j)\cos(\alpha_j) & -\cos(\theta_j)\sin(\alpha_j) & r_j\sin(\theta_j) \\ 0 & \sin(\alpha_j) & \cos(\alpha_j) & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Once these parameters and frames are defined, the pose or position of the end effector with respect to global can be found using the following pose algebra equation, equation (2) [1, p. 199].

(2)

$$\xi_1 \oplus \xi_2^1 \oplus \xi_3^2 \oplus \xi_4^3 \oplus \xi_e^4 = \xi_e$$

Once this is computed the $x, y,$ and z coordinates of the end effector will be extracted from this final pose. This can be used along with the $x, y,$ and z coordinates of the hand to determine the error, which will be minimized using inverse kinematics. The next step will be to numerically solve for the angles needed to move the end effector in a way that will minimize this error. This is done by computing the equation below 5000 times to find joint displacements that will successfully bring the end effector to the hand.

(3)

$$q(n+1) = q(n) + T_s \cdot J^+ \cdot k \cdot error$$

Equation (3) estimates where joint positions will need to be one timestep in the future. When this is computed over and over again, where each time the new joint positions are based on the previously calculated ones, it will eventually converge at a desired task space. In this case, the task space consists of the end effector reaching the hand. In equation (3), J^+ is the pseudo-inverse Jacobian matrix, which depends on the partial derivatives of the $x, y,$ and z coordinate equations [1, p.229]. The error is the location of the hand with respect to global minus the location of the end effector with respect to global. $q(n)$ is the previously calculated angles, and k is a proportionality constant that is derived through experimentation. Using this method, a MATLAB® function was created that takes in the current angles of the robot's joints and the $x, y,$ and z coordinates of the hand and computes the needed angles through this iterative process.

4. Implementation

4.1. Step 1: Segment the Image Provided by the Camera

The first step in this process is to take a picture or in this case one frame of a video, from the embedded RGB D camera on the end of the robotic arm and segment this image. Using the threshold-based image segmentation method a new image is created where every pixel that has a red value 1.2 times the size of the green value is equal to one and all other pixels are set equal to 0.

Figure 6. Image of hand taken with the camera and its segmented image. The Image on the right is the RGB image of the hand. The image on the left is the black and white segmented image.

4.2. Step 2: Detect Hand

Once the segmented image is created an average of all the values in this binary matrix is calculated, if that value is above a certain threshold, a hand is considered present, if not, a hand will not be detected. This ensures that if there is not a hand present and the background has a small defect where the image segmentation method recognizes pixels with this red-to-green value ratio, it does not think that there is a hand. In this particular case, the threshold value used was 0.1, meaning that if less than 10% of the image was considered a hand, a hand was not found.

4.3. Step 3: Locate the Center of Mass of the Hand

The 2D center of mass of the hand is computed by finding the center of the white pixels, or values of 1 in the segmented image. In this calculation, it was assumed that all pixels would be weighted equally, or have an equal mass of 1. Equation (4) is then used to find the x and y of the center of mass. The values will be used later in the calculation of the pose of the hand with respect to the camera.

$$x_c = \frac{\sum_{n=1}^{\infty} M_n x_n}{\sum_{n=1}^{\infty} M} \qquad y_c = \frac{\sum_{n=1}^{\infty} M_n y_n}{\sum_{n=1}^{\infty} M} \tag{4}$$

Due to the limitations of the depth camera on the robot, calculating an accurate z coordinate for the hand was not possible. This process simply sets the z coordinate of the hand to a constant value of 0.25m. This value lies just past the end effector, ensuring that the end effector will move forward just a small amount for every picture taken, still allowing the robot to move forward in the desired direction, but also avoiding the inconsistent values provided by the depth camera.

4.4. Step 4: Find the Global Pose of the Hand

The global x y and z coordinates are calculated for the hand by finding the pose of the hand with respect to the end effector. Once this is computed the pose of the end effector with respect to global will be calculated using forward kinematics. Using the pose of the end effector with respect to global and the pose of the hand with respect to the end effector the pose of the hand with respect to global can be calculated. This is done for each image taken, or each frame of the video feed provided by the camera. This process is done within a user-defined MATLAB® function titled FindHandLocation.

4.5. Step 5: Inverse Kinematics Function

The final step is to read the current angles of the robotic arm joints and use the previously calculated position of the hand to determine how much the joints must be rotated to reach the hand. This is done through the use of the inverse kinematics function that calculates the desired joint positions to reach the hand by minimizing the error, or the difference between where the hand is and where the end effector is. This process will be repeated for every image taken by the camera, allowing the end effector to not only go to the location of the hand but also follow the hand as it moves. This is done within the user-defined MATLAB® function titled CameraIK.

4.6. Simulink® Model

This system was modeled and simulated using Simulink®. The Quanser QArm has specific Simulink® blocks that must be present within the model to interface with the control system embedded in the robot. These can be seen in the model below along with the user-defined MATLAB® functions and blocks that perform the image processing and inverse kinematics.

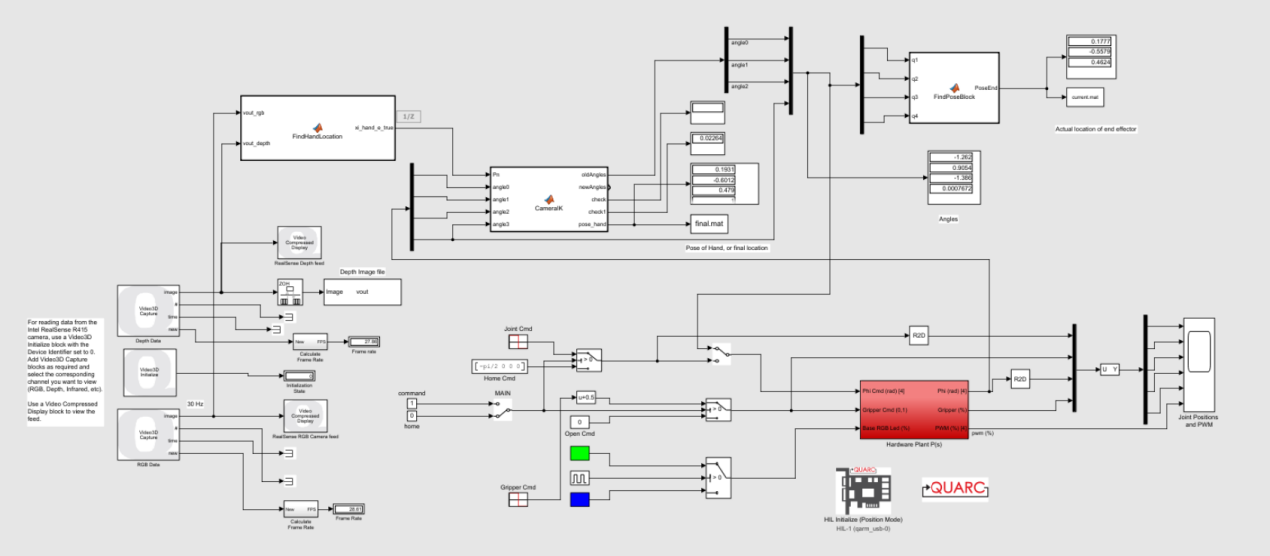


Figure 7. Image of the Simulink® model used to control the robotic arm. This model includes the blocks required for the Quanser QArm as well as the blocks defined for finding the center of mass of the hand and performing inverse kinematics.

5. Results

5.1. Image Segmentation Results

The hand detection process implemented a threshold-based image segmentation method where the pixels containing red values 1.2 times the size of the green values were considered a hand. This method was tested on 113 different images of hands in front of white backgrounds. Of the 113, there were only two images where a hand was present but not detected. In these cases, the general outline of the hand was segmented, however, since a hand is only considered present if 10% of the image has been labeled a hand, the algorithm did not detect the hand. In these two cases, the mid-section of the hand had much higher green values than the other images, therefore, the ratio of 1.2 between the red and green values was no longer valid. This increased green value was due to a significantly lighter skin tone and bright, almost unnatural lighting. Therefore, these two cases were not considered a concern in the process since the lighting was significantly different in these images than it would have been in the environment of the robot.

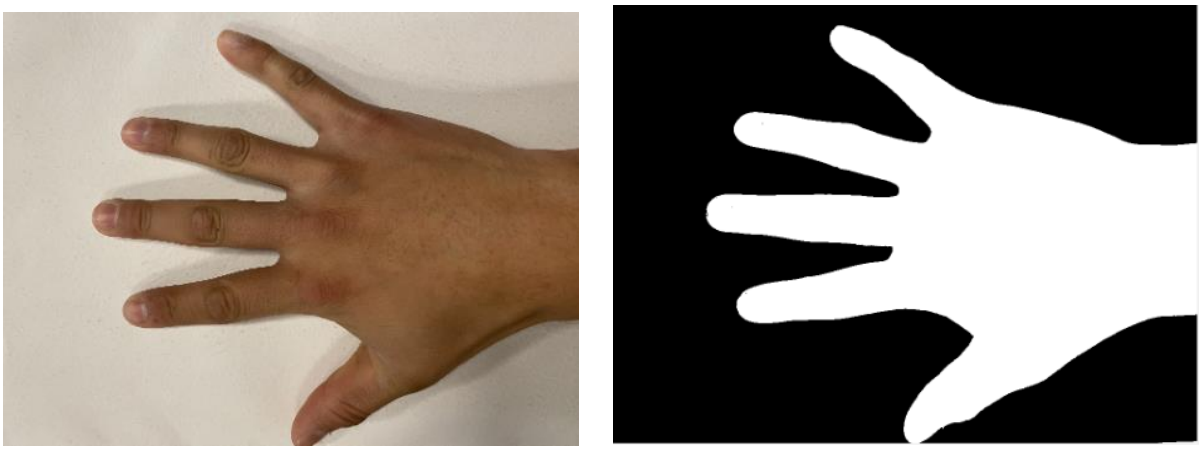


Figure 8. Image of successfully detected hand. The image on the left is the RGB image of hand. The image on the right is the segmented image. In this case, almost the entire hand is recognized as a hand and cleanly segmented from the background.

Figure 9. Image of unsuccessfully detected hand. The image on the left is the RGB image of the hand. The image on the right is the segmented image of the hand. In this case, only the edges of the hand were detected, and therefore only the edges of the hand were segmented properly. This caused a hand to not be detected since less than 10% of the image was considered a hand.

The hand detection method proved to have 98.2% accuracy in detecting a hand and properly segmenting the image into the hand and background.

5.2. Locating Center of Mass Results

In terms of the center of mass calculation accuracy, two different tests were examined. The first was whether the center of mass was found to lie within the bounds of where the image segmenting algorithm labeled a hand (within the white portion of the segmented image). The second test was whether or not the calculated center of mass was within the actual bounds of the hand (somewhere between or on the wrist and fingertips). Of the 111 images that properly segmented the hand, 110 of them found the center of mass to be within the white portion of the segmented image. Therefore, this method had an accuracy of 99.1% for the first test. For the second test, of the 111 properly segmented images 106 found the center of mass to lie within the actual bounds of the hand, having an accuracy of 95.5% for the second test.

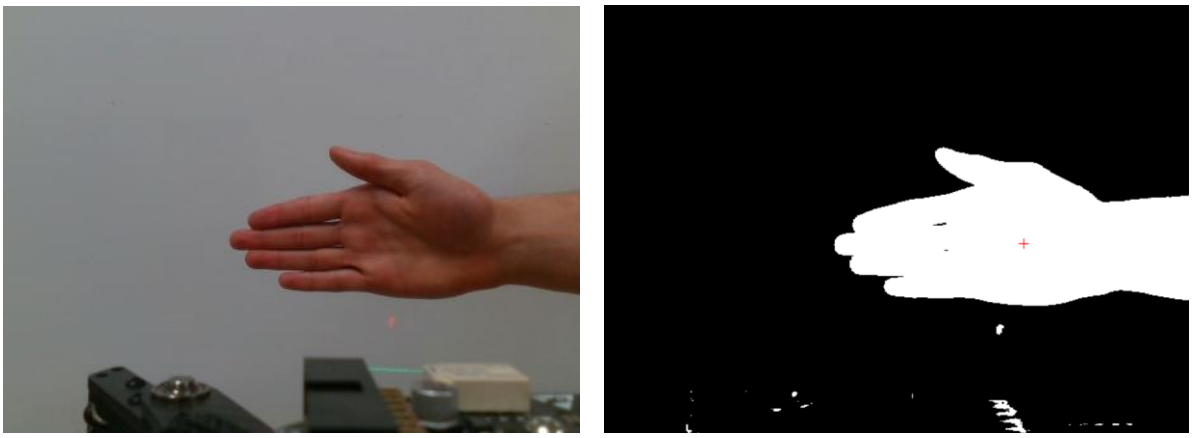


Figure 10. Image of Hand and its center of mass location that passed both tests. The image of the left is the RGB image of the hand. The image on the right is the segmented image of the hand with a red ‘x’ marking the calculated center of mass. In this case, the center of mass was calculated to be in the palm of the hand.

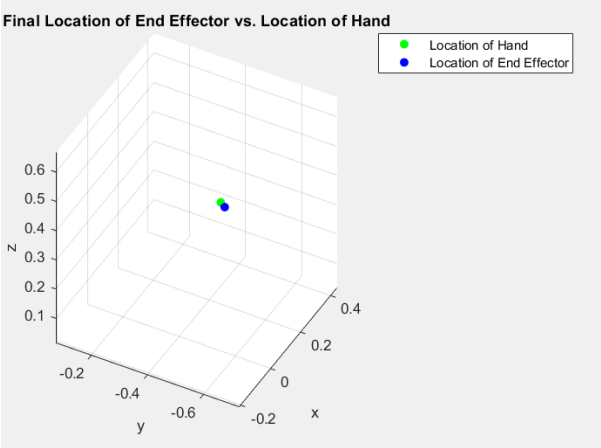
Figure 11. Image of hand and center of mass that passed test 1.The image on the left is the RGB image of the hand. The image of the right is the segmented image of the hand with a red ‘x’ marking the calculated center of mass. In this image, the forearm was also considered part of the hand. In this case, the center of mass was calculated to be on the person’s forearm instead of being on the actual hand.

Figure 12. Image of hand that did not pass either test. The image on the left is the RGB image of the hand. The image on the right is the segmented image of the hand where a red ‘x’ marks the calculated center of mass of the hand. In this case, some of the arm of the person was considered the hand. The center of mass was calculated to be on their wrist, where they were wearing a dark blue watch that was not considered part of the hand.

5.3. Inverse Kinematics

To determine the accuracy of the inverse kinematics process the location of the hand and the trajectory of the end effector were monitored. The final location of the end effector was compared with the average location of the hand in the case where the hand was held still. The magnitude of the final error between the hand and end effector location was computed for each run, there were 20 total runs. The success of reaching the hand was based on the calculated magnitude of the error. 14 of the 20, 70%, measured a final error magnitude less than 0.05m. 90% of the runs measured a final error magnitude less than 0.08m. And 100% of the runs measured a final error magnitude less than 0.1m. The gripper on the end of the arm reaches roughly 0.12 meters past the end effector. Therefore, in all of these cases, the gripper was within reach of the hand.

Below is a graph displaying the average location of the hand in the 3D space and the trajectory of the end effector. As can be seen, the end effector moves in the direction of the hand. Below is also a graph of the final location of the end effector and the average location of the hand.



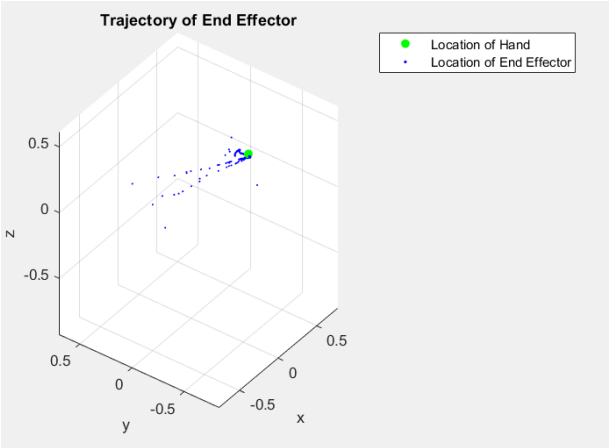


Figure 13. Final location of the end effector and its trajectory vs. location of the hand. The figure on the left shows the final location of the end effector in blue and the location of the hand in green. As can be seen they are very close together, however not exactly in the same spot. The image on the right shows the trajectory of the location of the end effector in blue and the location of the hand in green. As can be seen the end effector does move in the direction of the hand. This test had a final error magnitude of 0.048 meters.

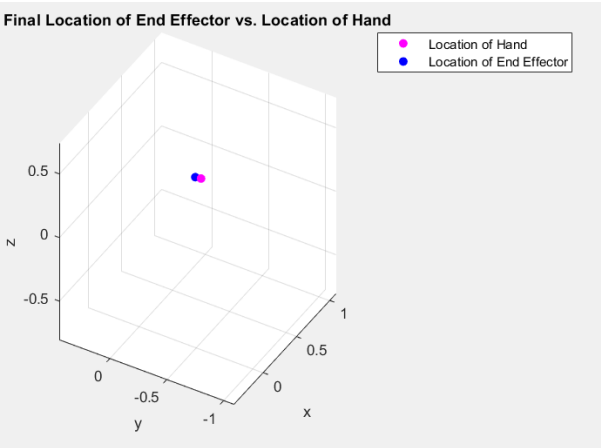


Figure 14. Final location of the end effector and its trajectory vs. location of the hand. The figure on the left shows the final location of the end effector in blue and the location of the hand in magenta. As can be seen they are very close together, however not exactly in the same spot. The image on the right shows the trajectory of the location of the end effector in blue and the location of the hand in magenta. As can be seen the end effector does move in the direction of the hand. This test had a final error magnitude of 0.018 meters.

6. Conclusion

This project uses a threshold-based image segmentation method along with numerical inverse kinematics to successfully control a robotic arm following a human hand. The image segmentation technique proved to have an accuracy of 98.2% in detecting a hand and a 95.5% accuracy in finding a reliable center of mass for the hand. There are several areas where this project and approach could be improved. In the image segmentation process having a white background was required to successfully see and detect a hand. This does provide limitations on the environment where this robotic arm could be used for this purpose. Using a more robust image segmentation method, or even introducing concepts of neural networks could provide an object detection algorithm that would be valid in a wider range of environments. In terms of the kinematics and motion of the robotic arm itself, there were limitations to the robot's function due to the nature of the RGB D camera. The camera is not able to obtain accurate depth readings when an object is within 0.16m of the camera. This means that when an object is close enough for the robot to stop or even grab something, the robot is not aware of it. Therefore, this robot can successfully follow objects and locate them. However, it cannot successfully stop and pick up or perform another action on an object when implementing computer vision with its embedded camera. Despite these limitations and areas of possible improvement, this project successfully demonstrates the use of computer vision-based control and the implementation of numerical inverse kinematics with a robotic arm system.

7. Acknowledgements

I would like to express my sincere gratitude to the engineering department at UNC Asheville who made this project possible through providing the necessary resources and equipment. I would also specifically like to thank Dr. Eli Buckner for his patience, guidance, and support, without it this project would not have been possible.

References:

[1] P. I. Corke, *Robotics, Vision and Control: fundamental algorithms in MATLAB®*. Cham, Switzerland: Springer, 2017.

[2] “Qarm,” Quanser. https://www.quanser.com/products/qarm/#overview. (accessed: Mar. 25, 2023)

[3] “Intel® realsense™ depth camera D415,” Intel RealSense. https://store.intelrealsense.com/buy-intel-realsense-depth-camera-d415.html. (accessed Mar. 26, 2023)

[4] “Intel® realsense™ depth camera D415 product specifications,” Intel.com. https://ark.intel.com/content/www/us/en/ark/products/128256/intel-realsense-depth-camera-d415.html. (accessed: Mar. 25, 2023).

[5] “Matlab,” MathWorks. https://www.mathworks.com/products/matlab.html. (accessed Mar. 29, 2023)

[6] “Simulink,” MathWorks. https://www.mathworks.com/products/simulink.html. (accessed Mar. 29, 2023)

[7] “What Is Inverse Kinematics?,” MathWorks. https://www.mathworks.com/discovery/inverse-kinematics.html#:~:text=Inverse%20kinematics%20is%20the%20use. (accessed Mar. 19, 2023)