Introduction to Robotics: Final Project

Collin Cutler, Daniel Parker

***Abstract*—In this project, we develop software enabling the QArm robot to pick up a tagged object and place it on a drop-off location. This is accomplished using April-Tags, which allows the localization of an object with respect to a camera's location. Using this information we can then transform the location of the object to the Base frame. This allows us to move the arm to pick up the object and place it on the drop-off location. Problems were encountered when implementing the readAprilTag() function into the Simulink Model. This was resolved by creating an external Matlab script that only performed the readAprilTag() function. The location of the object and drop-off were then inputed into the Simulink model where the QArm performed the pick and place automatically. Repeatability of the project was acceptable, though improvements to AprilTag detection and location can be improved.**

I. Introduction

A. Overview

Robotics have used cameras to detect motion and target locations for decades. In recent years, augmented reality research has resulted in technology that approximates 3D location and orientation using a single camera. One implementation of this technology is AprilTag. The robotic arm used in this project is a Quanser QArm four degree of freedom serial manipulator. It's comprised of a base, shoulder, elbow, and wrist joints. Figure 1 shows the QArm robot.

B. Project Purpose

The purpose of this project is to perform a pick and place of an object to a drop-off location. The location of the object and drop-off will be determined automatically by the camera of the QArm detecting AprilTags stuck to the object and drop-off location. Once the object and drop-off locations are determined, the QArm will then perform the pick and place automatically. In previous labs, motion planning was performed by moving the QArm end-effector to locations and creating way-points. These way-points were then executed by the QArm using software provided by the professor. This project will utilize its own process of motion planning.

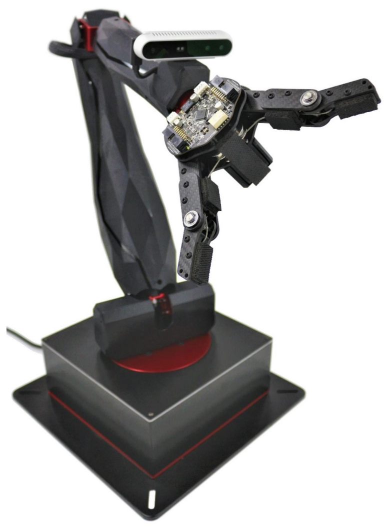

Fig. 1. Quanser QArm Robot

II. Design and Development

A. Technologies

1) AprilTag: AprilTags are QR-like codes that allow a calibrated camera to localize a known tag in an image relative to the camera. They do this by being unique orientation asymmetric patterns that are easy for a camera to recognize and orient. Knowing the real-world size of the tag they can then find the location and orientation of the tag relative to a camera. [1] [2]

B. Design

In general, the design of the Simulink model follows the flow chart in Figure 2. We used a main control switch to command the home position(input 1), camera position(input 2), or execute the pick-and-place motion(input 3). The pick-and-place motion was controlled by a counter that would start when the input to the control switch was 3 and would then count from 1 to 6 for each of the different states the arm should be in. This sequence can be seen in Figure 2 in the Counter Sequence sub-block where the numbers on the transitions indicate which position and gripper state is chosen for each value of the counter.

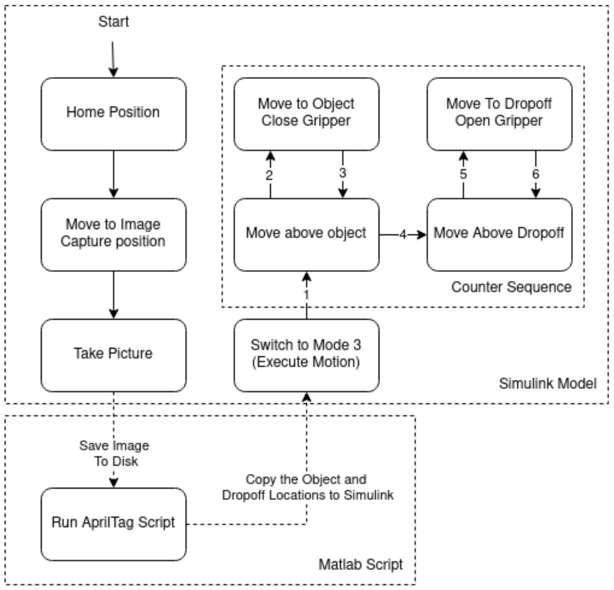
Start
Home Position
Move to Image Capture position
Take Picture
Move to Object Close Gripper
Move To Dropoff Open Gripper
Move above object
Move Above Dropoff
Counter Sequence
Switch to Mode 3 (Execute Motion)
Simulink Model
Save Image To Disk
Run AprilTag Script
Copy the Object and Dropoff Locations to Simulink
Matlab Script
1
2
3
4
5
6

Fig. 2. Model Flowchart

C. Matrix Mathematics

Using Figure 3, the homogeneous transformation matrix (HTM) of the base to the ArilTag can be found by multiplying intermediate transformation matrices together as shown in equation 1, where ${}^{base}_{w}T$ is the HTM of the wrist with respect to the base, and ${}^{w}_{cam}T$ is the HTM of the camera with respect to the wrist, and ${}^{cam}_{tag}T$.

$$ {}^{base}_{tag}T = {}^{base}_{w}T \times {}^{w}_{cam}T \times {}^{cam}_{tag}T \qquad (1)$$

The AprilTag's position, ${}^{base}_{tag}P$ is defined as elements (1:3,4) of ${}^{base}_{tag}T$. Once the position of the AprilTag with relation to the base is determined, inverse kinematics can be used to calculate possible solutions the QArm can take to pick up the object and place it at the drop-off location. For this project, the first set of solutions was used in the motion of the QArm.

III. Implementation

The approach to this project can be broken into a couple of different main parts. First, we need to find the location of both the object to pick up and the location where the object should be placed. Once the locations are found they need to be transformed from the camera coordinate frame to the base frame of the robot arm. Finally, we have all the information that the robot arm needs to move to the object, pick it up, move to the drop-off location, and put the object down.

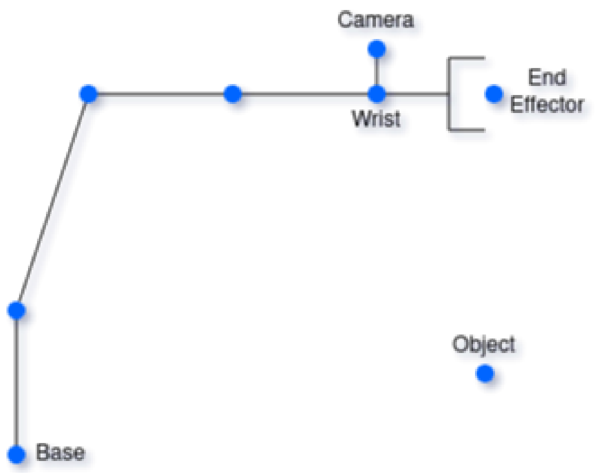

Fig. 3. QArm Coordinate System

A. Localization and Transformation

For our project, the main focus and innovation was the fact that we used AprilTags to allow dynamic object and drop-off locations. We were unable to get the AprilTags working within the Simulink code but could run it in a separate Matlab script. This was because the code to identify and localize the AprilTags was too slow and seemed to be a blocking call which interrupted the communication of Simulink with the arm when used inside the Simulink Model. To get around this we used the Simulink model to save a picture to disk, as shown in Figure 4, then manually ran the Matlab script to determine the locations of the Object and Drop-off point. The Matlab Script then called the readAprilTag() function to retrieve the pose of each of the tags in its field of view along with an ID of the tag that matches the specific pattern of the tag. These IDs are then matched to the known values we are looking for to the Object's tag ID and the Drop-off point's tag ID. The final step for localization is that the user manually copies the locations from the Matlab Output and puts them in their respective input boxes in Simulink. The locations are with respect to the camera's frame.

In order to transform the locations of the object and dropoff point from the camera's coordinate frame to the base frame we first transformed the locations to the wrist frame and then used forward kinematics to determine the transform from the wrist to the base. This allowed us to use the locations of the object and dropoff point relative to the base to then use inverse kinematics to determine what

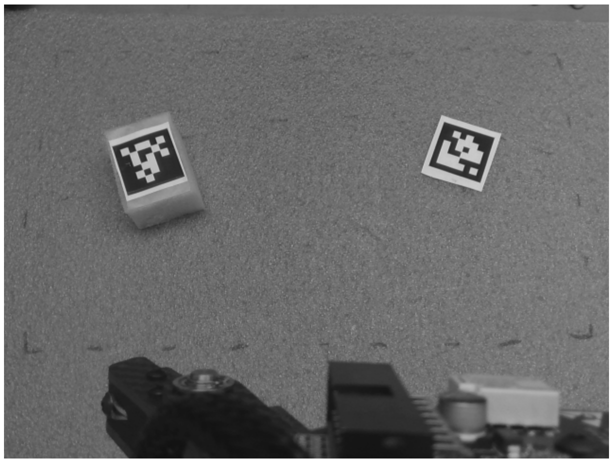

Fig. 4. QArm Camera Input Picture with Object (left) and Drop-off (right)

commands to send the robot to reach the locations commanded.

B. Motion Planning

Once the locations of the object and the drop-off point are inserted the model can then use them to pick up the object, move to the drop-off location, and put the object down. This was achieved by using a counter which started when the model was put into the Execute state(Value 3 at the top) that controls which location the arm is supposed to be at. It also sets the state of the gripper by switching the input to the corresponding state. The gripper itself is on a 1-second delay to allow the arm to reach the commanded position before it changes to allow the arm to reach the object and the dropoff point before closing or opening. This is done in a sequence of going to the location above the object, moving to the object and closing the gripper, moving back up, moving above the goal, moving to the goal and opening the gripper, and finally moving back up. These locations were calculated by taking the object position and the dropoff position in the base frame and for the locations above each location adding an offset.

IV. Results

When testing the developed Simulink model and Matlab script we were able to get everything generally working. We were able to capture the image (Fig IV), extract the tag poses, transform them into the base frame, and then pick-up (Fig IV) and dropoff the object (Fig IV) from those locations. This all went relatively smoothly with only some minor fine-tuning of the location of the camera relative to the wrist since the initial location was an estimate with rulers as there was no official documentation on the location of the camera relative to the wrist that we could find. Once this was tuned the model worked pretty well with the added restriction of avoiding placing the tags at the extreme edges of the image as that could cause errors in the pose extraction. The locations of the tags in Figure 4 are pretty close to as far left and right as we found work while they could be a bit further and closer while still extracting the pose correctly and being able to reach it with the arm.

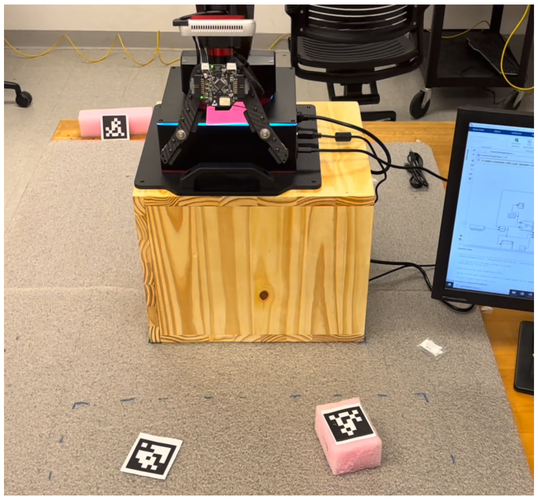

Fig. 5. Taking the picture for extracting the tag poses

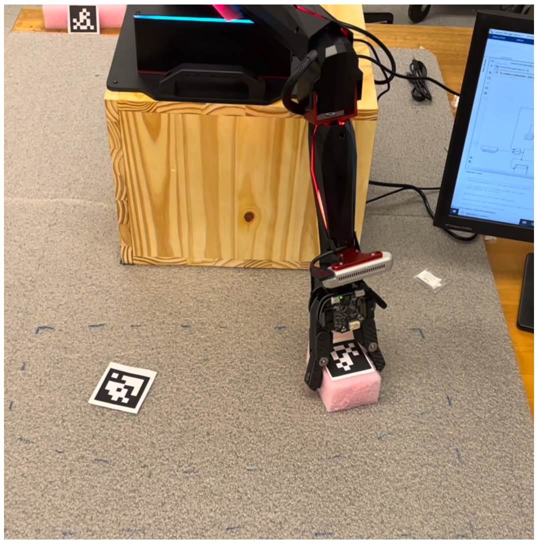

Fig. 6. Picking up the object

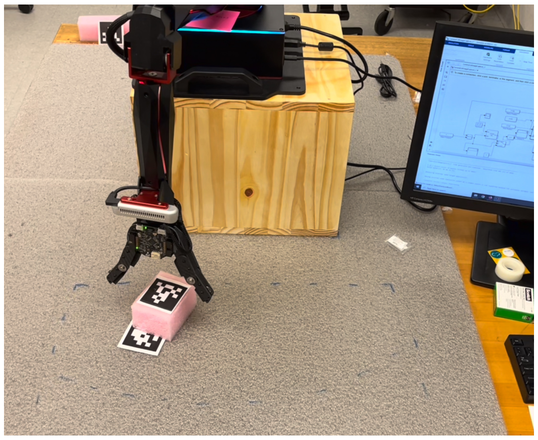

Fig. 7. Dropping off the object

V. Conclusion

In general, we were able to accomplish most of the goals of this project. We were able to detect the AprilTags, convert their locations into the base frame, and pick up and drop off the object at the marked location. There were a couple of snags that we ran into during the project but we were able to accomplish the overall goals despite them. We were able to use the tags to identify the locations of the object and dropoff point and use them to gather the locations of each for pick-up and dropoff.

References

[1] Mathworks. (2023) Detect and estimate pose for apriltag in image. [Online]. Available: https://www.mathworks.com/help/vision/ref/readapriltag.html

[2] T. R. of The University of Michigan. (2023) Apriltag-imgs. [Online]. Available: https://github.com/AprilRobotics/apriltag

VI. Appendix

Figure 8 shows the Simulink model used to control the QArm Robot. The following sections are the Matlab Scrips used by the Simulink Model:

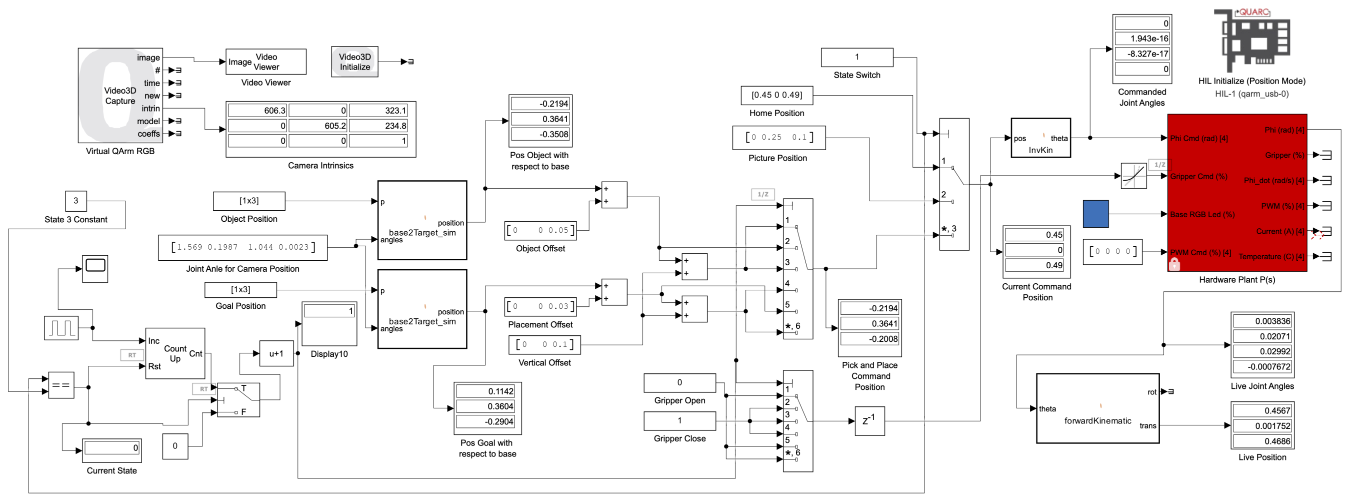

Fig. 8. Simulink Model

A. base2Target.m

```
function p = joint2Target(P_ct,jointAngles)
% This takes in the joint angles of the robot and position of the
    ↪ target
% with respect to the camera. The output is the transformation matrix
    ↪ of
% the target with respect to the base. ie where we want the end-
    ↪ effector to
% go.

% Define individual angles
t1 = jointAngles(1);
t2 = jointAngles(2);
t3 = jointAngles(3);
t4 = jointAngles(4);

T_0w = FwdKinQARM_wrist(t1,t2,t3,t4); % base to wrist
T_wc = [0, 1, 0, .05;
     -1, 0, 0, -0.05;
      0, 0, 1, -0.03;
      0, 0, 0, 1]; % wrist to camera

T_ct = [1,0,0,P_ct(1);
      0,1,0,P_ct(2);
      0,0,1,P_ct(3);
         0,0,0,1]; % camera to target
T_0t = T_0w*T_wc*T_ct;
```

```
p = T_0t(1:3,4)';
end





%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function T0_w = FwdKinQARM_wrist(t1,t2,t3,t4)
%FwdKinQARM creates the transformation matrix for the base to the tool
   ↪ frame
% Inputs angles in radians of each joint

% QARM lengths specified in the Lab 6 coordinate system
L1 = 0.14; %m
L2 = 0.35; %m
L3 = 0.05; %m
L4 = 0.25; %m
L5 = 0.15; %m

% Angle between vertical and common perpendicular of joint 2&3
beta = atan2(L3,L2);

% DH Table of QARM
DH = [0,0,L1,t1;
   -pi/2,0,0,t2-pi/2+beta;
   0,sqrt(L2^2+L3^2),0,t3-beta;
   -pi/2,0,L4,t4;
   0,0,L5,0];

% Calculates Transformation matrix base to wrist frame.
T0_w = eye(4);
for i= 1:4;
   T0_w = T0_w*HTM(DH(i,1),DH(i,2),DH(i,3),DH(i,4));
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function homoTransferMatrix = HTM(alphai_1,ai_1,di,thetai)
%homoTransform This function creates the Homogeneous Transformation
   ↪ Matrix
% The angles are in radians. Use deg2rad for conversion.
homoTransferMatrix = [cos(thetai),-sin(thetai), 0,ai_1;
   sin(thetai)*cos(alphai_1), cos(thetai)*cos(alphai_1),-sin(alphai_1)
      ↪ ,-sin(alphai_1)*di;
   sin(thetai)*sin(alphai_1),cos(thetai)*sin(alphai_1),cos(alphai_1),
      ↪ cos(alphai_1)*di;
   0,0,0,1];
end
```

B. forwardKinematics.m

```
function transform = forwardKinematics(theta)
   theta1 = theta(1);
   theta2 = theta(2);
   theta3 = theta(3);
   theta4 = theta(4);
   l1 = 0.14;
   l2 = 0.35;
   l3 = 0.05;
   l4 = 0.25;
   l5 = 0.15;
   l_23 = sqrt(l2 * l2 + l3 * l3);
   beta = atan2(l3, l2);

   % theta, alpha, a, d
   dh = [theta1, 0, 0, l1;
        theta2 - pi/2 + beta , -pi/2, 0, 0;
        theta3 - beta, 0, l_23, 0;
        theta4, -pi/2, 0, l4;
            0, 0, 0, l5];
   % Create identity matrix of 4x4
   transform = eye(4);
   for i = 1:size(dh,1)
      % append("T_", char(i-1 + '0'), "_", char(i+'0'))
      t = homoTransMatrix(dh(i,1), dh(i,2), dh(i,3), dh(i,4));
      transform = transform * t;
   end
   rot = transform(1:3, 1:3);
   trans = transform(1:3, 4);
```

C. homoTransform.m

```
function transform = homoTransform(alpha,a, d, theta)
  transform = [cos(theta) -sin(theta) 0 a
            sin(theta)*cos(alpha) cos(theta)*cos(alpha) -sin(alpha) -
               ↪ sin(alpha)*d
            sin(theta)*sin(alpha) cos(theta)*sin(alpha) cos(alpha) cos(
               ↪ alpha)*d
            0 0 0 1];
end
```

D. InverseHomogeneousMatrix.m

```
function T_inv = InverseHomogeneousMatrix(T)
   % Check if the input matrix is a 4x4 matrix
   if size(T) ~= [4, 4]
      error('Input␣matrix␣must␣be␣a␣4x4␣homogeneous␣transformation␣
         ↪ matrix.');
```

```
    end

    % Extract the rotation matrix and translation vector from the input
       ↪ matrix
    R = T(1:3, 1:3);
    p = T(1:3, 4);

    % Calculate the inverse of the rotation matrix
    R_inv = transpose(R);

    % Calculate the translation vector of the inverse
    p_inv = -R_inv * p;

    % Construct the inverse homogeneous transformation matrix
    T_inv = [R_inv, p_inv; 0, 0, 0, 1];
end
```

E. *InverseKinematic.m*

```
function sol1 = InverseKinematic(position)
%INVERSEKINEMATIC2 Summary of this function goes here
% Detailed explanation goes here
    px = position(1);
    py = position(2);
    pz = position(3);

    % Arm Dimensions
    l1 = 0.14;
    l2 = 0.35;
    l3 = 0.05;
    l23 = sqrt(l2^2 + l3^2);
    beta_arm = atan2(l3, l2);
    l4 = 0.25;
    l5 = 0.15;
    % Set final position equal to tip of end effector
    l4 = l4 + l5;

    % Theta1 Solutions
    theta1_1 = atan2(py, px);
    theta1_2 = atan2(-py, -px);

    % Theta4 Solution
    theta4 = 0;

    dist_between_T1_T5 = sqrt(px^2 + py^2 + (pz - l1)^2);
    % transform end effector position for the T01 so everthing is planar
       ↪  on
    % the XZ-plane then use law of cosines to find angles for triangle
       ↪ of
    % arm sections and distance between Base and end effector.

    T1Px_1 = InverseHomogeneousMatrix(homoTransform(0, 0, l1, theta1_1))
       ↪  * transpose([px py pz 1]);
    angle_1 = atan2(T1Px_1(1), T1Px_1(3));
    T1Px_2 = InverseHomogeneousMatrix(homoTransform(0, 0, l1, theta1_2))
       ↪  * transpose([px py pz 1]);
    angle_2 = atan2(T1Px_2(1), T1Px_2(3));
    % 3side Triangle solution law of cosines
    a = l23;
    b = l4;


    if T1Px_1(1) > T1Px_2(1)
       c = sqrt(T1Px_1(1)*T1Px_1(1) + T1Px_1(3)*T1Px_1(3));
       beta = acos((a^2 + c^2 - b^2)/(2*a*c));
       gamma = acos((a^2 +b^2 - c^2)/(2*a*b));

       theta2_1 = angle_1 - beta - beta_arm;
       theta3_1 = pi/2 - gamma + beta_arm;
       sol1 = [theta1_1, theta2_1, theta3_1, 0]';
       return
    else
       c = sqrt(T1Px_2(1)*T1Px_2(1) + T1Px_2(3)*T1Px_2(3));
       beta = acos((a^2 + c^2 - b^2)/(2*a*c));
       gamma = acos((a^2 +b^2 - c^2)/(2*a*b));
       theta2_1 = angle_2 - beta - beta_arm;
       theta3_1 = pi/2 - gamma + beta_arm;
       sol1 = [theta1_2, theta2_1, theta3_1, 0]';
       return
    end
end
```