Robotic Pick and Place System

Zeynep Seker
Robotics Engineering
Worcester Polytechnic Institute
Worcester,USA
zseker@wpi.edu

Connor Craigie
Robotics Engineering
Worcester Polytechnic Institute
Worcester,USA
cacraigie@wpi.edu

Peter Dentch
Robotics Engineering
Worcester Polytechnic Institute
Worcester,USA
pdentch@wpi.edu

***Abstract*— Forward and inverse kinematic functions are developed for the quintic trajectory planning for the RBE 3001 Robot Arm. Local objects are identified and manipulated using computer vision. The robot arm is programmed to identify the different types of objects and use its grippers to manipulate the object around the baseboard.**

I. INTRODUCTION

In order to manipulate the detected object on the work space, D-H parameters for the home position were established. Furthermore, functions for both forward and inverse kinematics were developed in order to plan a trajectory path for the end effector motion. The USB camera attached to the baseboard frame was registered to the robot's Nucleo Board. The forward kinematic transformation needed from the base frame to the camera was established by using a checkerboard as reference. Snapshots of different bulbs were retrieved in order to create colour masks for each bulb. The bulbs were detected as circles within a given radius range. After achieving colour detection, different sizes of bases were introduced to the work space. In order to distinguish the different base sizes, their areas were calculated and assigned a label.

II. METHODOLOGY

A. *Server Setup*

By reading the encoder values from the terminal, coordinates for the robot's home position were obtained. Within Eclipse, the given *main()* stored arbitrary values for the robot's home position. These three values represented the theta offsets required to assume the robot's home position. All the setpoints assigned in the future take the home position (-1048, -1045, 1970) as the "origin".

Next, a new server, *STATUS SERVER*, was created in order to constantly receive the current state of each joint. Afterwards, a PID Configuration Server was created to explicitly write new PID gains. These constants are sent through a single packet and returned for confirmation. Once received in Eclipse, the *setPIDConstants( )* is used to alter the gains of the robot.

B. *Forward Kinematics*

Denavit-Hartenberg parameters of each joint in the home position were calculated. (See Figure 1 and Table 1 for the detailed DH parameters and frame orientation). Each frame was selected in a way that the joint angles would always be represented as θn*.

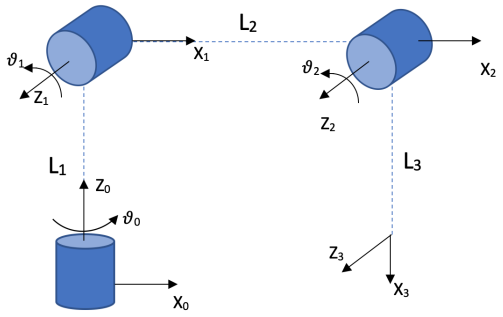

Fig. 1. DH Frames at Home Position

Frames	θ	d	a	α
1	-θ*$_1$	L_1	0	90^o
2	θ*$_2$	0	L_2	0
3	-90^o+θ*$_3$	0	L_3	0

Table 1:DH Parameters at Home Position

The Denavit-Hartenberg table above is used to develop the homogeneous transformation matrix from the base to the end effector. Using the intervals of the DH table, the intermittent frames were used to represent the robot links. From here, link locations are mapped in 3 dimensional Cartesian space. This is a great tool to compare the virtual output of the robot with the physical results.

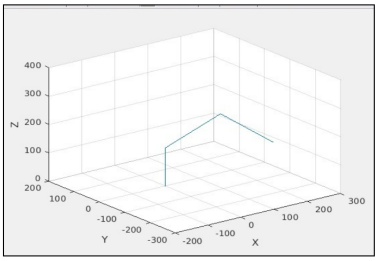

Fig. 2. Simplified Robot Links in Cartesian Space

Using the homogeneous transformation matrix, the three forward kinematic equations can be derived from the positional column. Equations relating Px, Py, and Pz to theta are produced in respective rows of the 3x1 positional column. The first use of the forward kinematic function *fwk3001* was to assign multiple points in the two dimensional plane, attempting to trace the vertices of a triangle. These points are not defined by their location in relation to the base frame, they are derived from the given theta displacements input into the forward kinematic equations. Using cubic trajectory generation, a path of multiple time depended setpoints are computed for each joint displacement. This calculated trajectory acts as a continuous function between two setpoints. The coefficients of this cubic function are calculated using the assumption that the final and initial velocities of the end effector should be zero. MATLAB is then used to calculate the remaining polynomial coefficients based on input parameters such as current time, final time, current position, and final position. This trajectory generation would eventually be modified to produce quintic set point generation. Quintic trajectory generation requires the parameterization of initial and final accelerations as well.

When all three cubic trajectories act synchronously, the motion of the robot becomes more fluid. In Figure 3, the theta displacements are mapped as a function of time. Also depicted is the resultant motion in the XZ plane produced by forward kinematics.

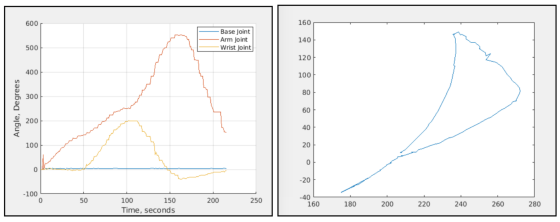

Fig. 3. Theta Displacements and Resultant Motion in XZ plane

C. *Inverse Kinematics*

The positional forward kinematic equations are used to solve for inverse kinematics. Inverse kinematics is incredibly important for the continuation of the project. When analyzing images for bulb locations, the algorithm will require a x-y-z positional input. Therefore, the forward kinematics must be rearranged such that all three joint displacements are in terms of the input x-y-z coordinate. These functions output the required theta values for all three joints in order to place the end effector at a given x-y-z point in Cartesian space. The figure below shows the three inverse kinematic equations that input x-y-z and output a value of theta. This value of theta will be linked to either the base, the arm, or the wrist of the robot.

$$\theta_1 = atan2(y, x);$$
$$\theta_2 = atan2((z - l_1), \sqrt{(x^2 + y^2)}) + acos(l_2^2 + x^2 + y^2 + ((z - l_1)^2 - l_3^2)(2 * l_2 * \sqrt{x^2 + y^2 + (z - l_1^2)}$$
$$\theta_3 = acos((l_2^2 + l_3^2) = (x^2 + y^2(z - l_1)^2))/(-2 * l_2 * l_3))) - 90^o$$

With the application of inverse kinematics, the robot is given a coordinate in the cartesian space. It then converts that coordinate into the three associated theta displacements. These displacements are then sent to the robot as its new setpoints. These set points can be interpolated to provide fluid motion. Without interpolation, the robot will attempt to move to the new position as fast as physically possible. Interpolation will produce time dependent set points between the robots current and desired position. Initially, a linear interpolation was applied between given setpoints. This produced evenly spaced intervals between each setpoint. However, in this case, the robot does not decelerate when approaching its desired location. To account for end effector acceleration, the robot uses the quintic setpoint generation in all three axes of motion. When parameterizing accelerations, the program will produce setpoints further apart as it accelerates away from its initial location, and then positions points closer together as it approaches its final position. This can be visualized below in Figure 5

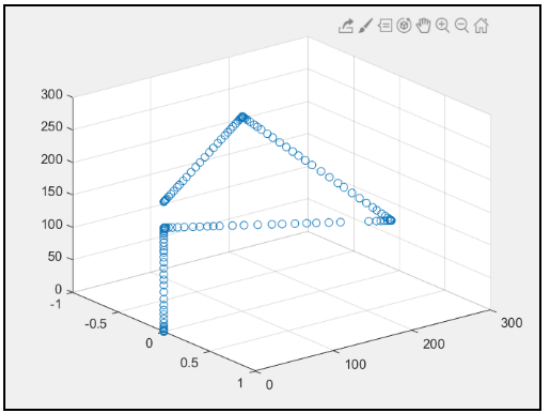

Fig. 4. Quintic Setpoint Generation in the Cartesian Space

Inverse kinematics can also be applied to the end effector velocities using the velocity Jacobian. The velocity Jacobian is the partial derivative of all three positional values with respect to the three identified degrees of freedom.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial C_1}{\partial q_1} & \frac{\partial C_1}{\partial q_2} & \cdots & \frac{\partial C_1}{\partial q_{3n}} \\ \frac{\partial C_2}{\partial q_1} & \frac{\partial C_2}{\partial q_2} & \cdots & \frac{\partial C_2}{\partial q_{3n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial C_m}{\partial q_1} & \frac{\partial C_m}{\partial q_2} & \cdots & \frac{\partial C_m}{\partial q_{3n}} \end{bmatrix}$$

Fig. 5. Generic Velocity Jacobian Matrix

The Jacobian is used to develop a relationship between the velocity of the joints and the instantaneous velocity of the end effector in the Cartesian coordinate system. The Jacobian is consistent for all orientations and positions of the robot in space. Therefore, it can be associated with a constant global variable, and used throughout the main script to compute the live velocities of the end effector. By taking the determinant of the velocity Jacobian matrix, stages of singularity can be checked. If the determinant reaches 0, then the robot arm

is at a perfect singularity. However, it is highly unlikely the robot will ever enter a perfectly singular state. Therefore a bounding range was created in a way that if the determinant approaches 0, then the arm returns to a safe homing position.

D. Relating Two Dimensional Camera Perspective to Cartesian Space

Using transformation matrices, the perspective of the camera can be related to three dimensional cartesian space. This is done using the given *points2world( )* function. The geometry of the camera space is known. Using this work space, the function assigns a theoretical third dimension bounding plane relative to the known space. Using previous camera calibrations with a checkerboard plane, the camera can understand the orientation of this 3D plane and determine its relative orientation. Therefore it can produce the homogeneous transformation between a fixed location on the checkerboard and the camera's perspective. This transformation will be stored as a constant in the code as *TCamToChecker*. This resultant transformation is just the first step in obtaining the overall transformation from the camera to the base frame.

$$\left(T_{Checkerboard}^{Camera}\right)\left(T_{Home\,Position}^{Checkerboard}\right)\left(T_{Base\,Frame}^{Home\,Position}\right) = \left(T_{Base\,Frame}^{Camera}\right)$$

Fig. 6. Construction of the Homogeneous Transformation from Camera to Base Frame

The overall transformation will be constructed from three separate transformation matrices. The first, is the previously determined *TCamToChecker*, the second is the *TCheckerToHome*, and the final is the *THomeToBase*. The *TCheckerToHome* is calculated from the fixed geometry of the board. The home position is defined as the location where the end effector rests when all theta setpoints are equal to zero. This was previously established in the project and will now be used as a fixed location to apply an intermediate frame. This transformation was then calculated using physical displacements measured on the board.

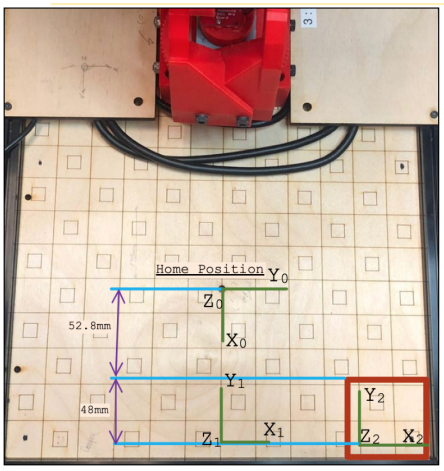

Fig. 7. Dimensional Analysis-Checkerboard to Home Position

The final transformation *THomeToBase*, can be calculated using the geometric definitions of the robot. The final transformation from camera to base frame will then be computed as the product of all three previously calculated transformations. All reference frames are fixed in space, therefore we can assume that this final homogeneous transformation matrix will be constant throughout the operation of the robot. This fixed matrix can then be used in the main function to repurpose data seen by the camera, and insert it into the reference of the robot.

E. Vision Processing and Data Acquisition

Using computer vision, the system is able to detect and discern between the different bulbs and sort them accordingly. Before a new trajectory is generated and sent to the robot, an image of the workspace is captured using the USB webcam. This image is initially filtered such that only the wooden base of the workspace remains. Any bulbs or other objects outside of this area will not be processed. A black and white, or "binary" image of the area of interest is displayed in Figure 8.

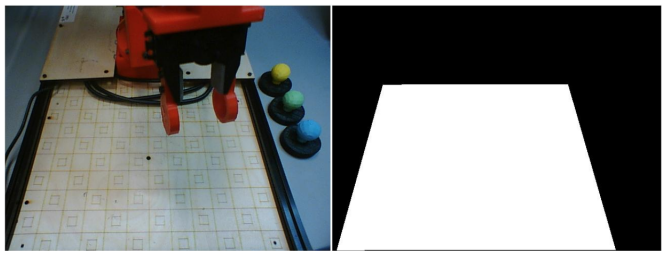

Fig. 8. Polygonal Binary Image Mask of the Workspace

Once this image is present, various color masks were applied to it in order to isolate only the yellow, green, and blue colors of any bulbs which could be present in the workspace. These masks were generated by the MATLAB vision processing toolkit which allowed us to finely tune what pixels in the CIELAB color space should remain in the image while the rest was filtered out. This created binary or black and white images which could then be passed to another function to detect circular contours. This ensures that the detected colors belong to the spherical bulbs and were not false positives. A square of 140 pixels was then drawn around the center of the detected bulb with the intent of discerning between a large base and a small one.

Fig. 9. Comparison of Masked Large and Small Bases

These square images are then filtered through a black color mask for detecting how much of the image is black. The detected black area of the base is then calculated within

the 140 pixel frame. If the detected area breaks the given threshold, the base is labeled as large. Otherwise the base is labeled as small. Some examples of these filtered contours are pictured in Figure 9.

After obtaining this information it is returned as an organized data structure containing the bulb centers and a boolean value to indicate the size of the base. This vital data in our program called *bulbParams* is a three by three matrix, organized by row for each bulb color detected. The columns of this matrix are the x position, y position, and base size being a value of one or negative one. Should a color bulb not be detected in the workspace of the image, the values of the x and y in the matrix are set to 999 in order to represent a null value meaning no bulb of this color is present. The *bulbParams* matrix is finally returned by the vision function and then used in generating the next trajectory for the arm. This informs the program which bulb is being picked up and thus where it should be sorted.

F. Rudimentary Program Description

The program exists in a constant looping state. Once powered on, the robot will forever attempt to detect objects to manipulate. If an object is not detected, the robot will be sent a one second trajectory to its current position. Therefore it will hold it position and check for updates in the workspace one second later. If an object is detected it will run through its object prioritization.

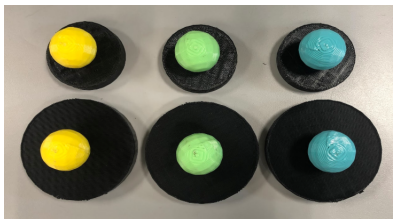

Fig. 10. Manipulation Objects

The system will first detect yellow bulbs in the workspace, then green, then blue. It will also always manipulate the bulbs in this order. This predetermined preference was an engineering design choice. By preferring certain bulbs over others, we can assure that all bulbs of one color are removed before continuing with the sorting process. Once a bulb is detected, the program will move the robot to a setpoint 80mm above the bulb. The system then checks again to assure the position of the bulb has not changed. However, if the position of the bulb has changed within a given interval of error, the robot will compensate and generate a new trajectory to close the gap. This acts as our dynamic tracking to assure the robot is positioned properly before attempting to grasp the bulb. Before lowering into gripping position, the robot takes one last picture of the bulb to assign its parameters in *bulbParams*. Once the robot moves to grasp a bulb, it enters a state machine that will determine where the bulb will be placed. Within the matrix *bulbParams* the data structure is designed in such a way that its components can be used within trajectory generation. Rather than having nested if statements and complicated functionality, the program uses the information within *bulbParams* to assign a new trajectory to the robot. The robot intends to sort large bases on the mirror side of the workspace from small bases. The robot is then expected to sort yellow, green, and blue in ascending order on their respective side based on base size.

TrajX =50 + (color numeric)(50)
TrajY = (base boolean)(200)

Equation1: Trajectory Generation Equation Using *bulbParams* Inputs

Due to the parametrization of color as scalar constants (yellow = 1 : green = 2 : blue = 3), and base size as boolean integers (large base = 1 : small base = -1), these scalars can be used in trajectory calculations. Therefore, as seen in Equation 1, X and Y displacements can be calculated for any combination of bulb color and base size. This allows for consistent organization.

III. RESULTS

By following the procedures detailed in the labs, the challenge of efficiently controlling the robotic arm manipulator was completed successfully. The robot was tasked with sorting bulbs according to color and base size. Fast packet communication with the server onboard the Nucelo processor was achieved. This fast transmission cycle provided quick commanding and receiving of state data from the arm. This data was used to log and visualize the arm joint angles and angular velocities to better comprehend how the robot is functioning in real time and to make necessary calculations. The mathematical theory detailed during lecture was then implemented in order to calculate the location of the arm's end effector; allowing for the visualization of the arm graphically and properly predicting its link locations in 3D space. Using forward kinematic positional equations, the inverse kinematic expressions were derived. Commanding the arm to various poses was implemented using this set of inverse kinematics to find the joint angles needed to drive it to various positions and orientations within its workspace. This allowed for the development of a helper function that controlled the arm. This helper function was then applied using MATLAB and computer vision. The algorithm used proved very robust in discerning between the three different bulb colors and sizes. This information allowed for finally controlling the arm to manipulate the bulbs from anywhere in the workspace and place them in sorted locations.

A demonstration of the final project can be found under the following link: https://youtu.be/-dSBT4T3I7o

IV. DISCUSSION

Throughout the project many obstacles have been faced. We had issues regarding the frame positions, thus the DH parameter signs. However, with the implementation of multiple test trials, we were able to achieve accurate positional outputs from our forward kinematic equations.

These positional outputs of the homogeneous transformation matrix were physically measured by the team. This would assure that the robot has landed in the correct location as verified by our calculations.

It was also found that the encoder values of the base joint were inverted. This caused a mirroring of values about the x-z plane, resulting in an inversion of all y-values. This issue was discovered when the inverse kinematic equations drove the end effector to inconsistent locations. The resulting issue was discovered quickly; it was clear that the robot's position was mirrored. We assume this issue is derived from the physical orientation of the sensor built into the robotic arm.

While comparing the different sizes of bases, our initial strategy was to use the function imfindcircles to compare the bases' radii. However, this did not result in accurate comparison. So, we solved the challenge by comparing the areas of the bases. By looking at the black area around a bulb we were able to distinguish whether it was a large base or a small base.

While working to implement the vision function of our program, we noticed errors regarding false negatives of detecting bulb bases. Due to poor lighting, more black was recorded by the masking function. This caused the function to inconsistently detect small bases as large. This issue was resolved by using a lamp provided in the lab to brighten the workspace and make a more clear image for processing.

The procedures we performed as team members greatly exercised our creativity. To succeed we had to exhibit professional level skills in program development and problem solving. The course prepared us with all the appropriate knowledge to take on the task of automating a pick and place robotic system. Overall the experience of the Unified Robotics 3001 course was incredibly beneficial to our growth as robotics engineers.