Four Degree of Freedom Articulated Arm Control

Zachary Serocki
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
zdserocki@wpi.edu

Sarah Listzwan
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
salistzwan@wpi.edu

Alexander Kraemling
Dept. of Robotics Engineering
Worcester Polytechnic Institute
Worcester, United States of America
ajkraemling@wpi.edu

***Abstract*—Control for a four degree of freedom articulated arm was implemented to allow the robot to detect and sort five different colored balls. Forward Kinematics was implemented to determine the position of the end effector of the robot in task space. Differential Kinematics was then implemented to allow for conversion between joint velocities and end effector velocities. Inverse Kinematics was implemented to determine the joint angles required to reach a point Task Space. Trajectory planning functions were created to allow for quintic task space trajectory planning. Camera Vision was then implemented to allow the robot to recognize and differentiate between colored balls in the task space. A state machine was then implemented to determine the location of the objects, generate a trajectory to an object, move it to the correct color bin, and return to grab the next object.**

***Index Terms*—forward kinematics, inverse kinematics, articulated arm, differential kinematics, computer vision**

Introduction

A. Background and Motivation

To implement color sorting of objects using a four degree of freedom (4-DOF) articulated arm, layers of control systems were constructed. To begin the robot controller was capable of writing positions in degrees, or velocities in degrees per second to each of its joints. The robot contains four rotational joints. The robot was controlled using the DYNAMIXEL SDK with MATLAB.

On top of this base control layer, forward kinematics, differential kinematics, inverse kinematics, and trajectory planning were implemented to allow for task space control of the end effector of the robot. To allow for object sorting, the robot must be capable of recognizing and determining the position of different colored objects. A camera was trained on the checkerboard base and five different colored objects (red, orange, yellow, green, and blue). The checkerboard was used to determine the real world position of the objects.

To achieve the goal, these techniques were integrated to allow for quintic task space trajectory generation from the end effectors original position to the goal object, and from the goal object to its respective collection bin.

Methods

1) Implement Forward Kinematics by determining Denavit-Hartenberg parameters, developing intermediate transformation matrices, and post multiplying to determine world to end effector transformation matrix.
2) Implement Differential Kinematics by solving for Jacobian matrix from Forward Kinematics transformation.
3) Implement Inverse Kinematics using a numerical Newton-Raphson approach.
4) Develop quintic trajectory generation functions in task space using inverse kinematics.
5) Train camera system to recognize five distinct colors, and transform location of object in pixel space to task space
6) Combine all previous steps into a state machine to recognize, grab, and deposit objects in their colored containers.

Results

B. Forward Kinematics

To determine the location in task space of the end effector of the robot using the joint angles of the robot, the forward kinematics of the robot was computed. As shown in Figure 1, the reference frames for the robot were defined using the Denavit-Hartenberg convention [1]. DH parameters were then determined for each link (See Table I).

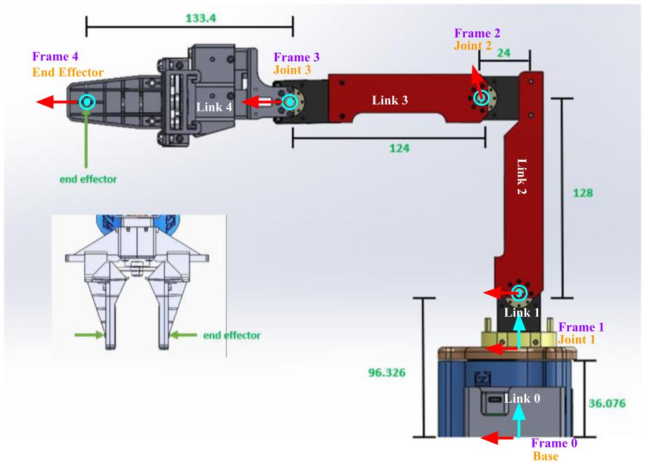

Fig. 1. Reference Frames of Robot

Link	θ (degrees)	d (mm)	a (mm)	α (degrees)
0	0	36.076	0	0
1	θ_1	96.326−36.076	0	-90
2	$\tan^{-1}\left(\frac{24}{128}\right) - 90 + \theta_2$	0	$\sqrt{24^2+128^2}$	0
3	$90 - \tan^{-1}\left(\frac{24}{128}\right) + \theta_3$	0	124	0
4	θ_4	0	133.4	0

TABLE I
DENAVIT-HARTENBERG PARAMETERS FOR EACH LINK

The DH parameters where then used to determine the intermediate transformation matrices using the a Z-axis rotation, Z-axis translation, X-axis translation, and X-axis rotation transformation matrix.

$$\begin{bmatrix} c_\theta & -s\theta c_\alpha & s_\theta s_\alpha & ac_\theta \\ s_\theta & c_\theta c_\alpha & -c\theta s_\alpha & as_\theta \\ 0 & s_\alpha & c_\alpha & d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

These intermediate transformation matrices were post multiplied to determine the base to end effector transformation matrix (See Figure 16). This was computed symbolically in MATLAB, then converted to a MATLAB function during startup. The joint angles can then be plugged in to determine the numerical transformation matrix for a given set of joint angles.

C. Differential Kinematics

To determine both the linear and angular velocities of the end effector given the joint angle velocities, the Jacobian matrix must be calculated.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J(\theta) \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix}$$

The Jacobian matrix is determined using the forward kinematics calculated above. The Jacobian is a 6 by n matrix with n representing the number of joints of the robot.

$$\begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} \\ [\hat{z}_1] & [\hat{z}_2] & [\hat{z}_3] & [\hat{z}_4] \end{bmatrix}$$

This matrix is precomputed symbolically on robot startup, and transferred to a MATLAB function for speed during use. The forward kinematic transformation matrix (See Figure 16, T_0^{EE}, is used to compute the upper half of the Jacobian by taking the partial derivative of the positional components with respect to each joint. The lower half of the Jacobian is computed by taking the $\hat{z}_i$ vector from their respective transformation matrices (See Figures 12-15) (ie. $\hat{z}_1$ is first three rows of the third column of T_0^1, and $\hat{z}_2$ is from T_0^2 and so on). The full symbolic Jacobian can be found in the Appendix, Figure 17.

To determine the joint velocities required to achieve an end effector scalar speed, the inverse of the Jacobian can be computed. controlling

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix} = J(\theta)^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

This allows for end effector speed control to be implemented. The speed of the end effector can be determine, and multiplied by the inverse of the Jacobian to determine the joint velocities needed to achieve that speed.

D. Inverse Kinematics

To determine the joint angles required for the robots end effector to reach a point in space, the inverse kinematics of the robot was computed. The Newton-Raphson root finding method was employed [2]. This method was chosen over an Algebraic method as it was found to be faster on average then the Algebraic inverse kinematic solution (See Figure 2).

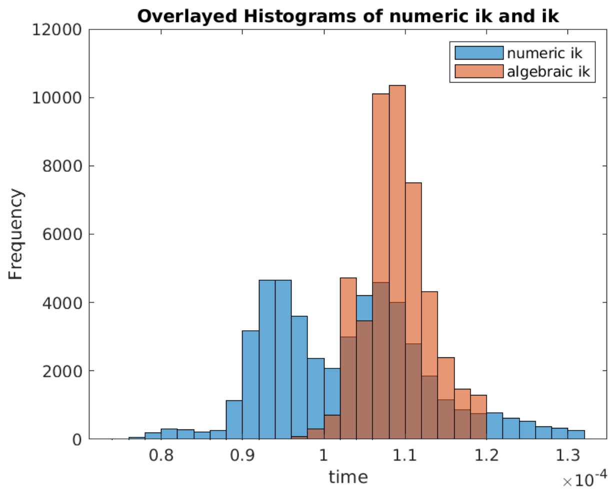

Fig. 2. Inverse Kinematic Algebraic vs. Numeric Compute Time

Numeric Inverse Kinematics was implemented by defining the error to be the difference between the ideal end effector position and the current end effector position (given a guess of the joint angles).

$$e(q_{guess}) = fk(q_{guess}) - p_{desired}$$

To determine the change in joint angles to approach the correct solution, the psuedoinverse of the Jacobian evaluated at the previous joint angle guess is used (psuedoinverse because the Jacobian is a non-square matrix). This is then multiplied by the error in the end effector to determine the change in joint angles.

$$\Delta q = J(q_{guess})^{\dagger} e(q_{guess})$$

The previous joint angles are then incremented by this change in joint angles.

$$q_{newguess} = q_{guess} + \Delta q$$

This process is then repeated until the error $e(q_{guess})$ is below 1 mm, or a cap of 50 iterations. Experimentally it was found that all solutions were found to converge within 50 iterations, given an initial guess of $q_{intial} = [0^o, 0^o, 0^o, 0^o]$.

E. Trajectory Generation

To allow for smooth movements between objects, a quintic trajectory function was created. A quintic trajectory is represented by a quintic polynomial where beginning and end angles, velocities, and accelerations are specified. A system of equations is solved to determine the quintic polynomial coefficients for each joint.

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5$$

$$q(t_0) = a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 + a_4 t_0^4 + a_5 t_0^5$$

$$q(t_f) = a_0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 + a_4 t_f^4 + a_5 t_f^5$$

$$v(t_0) = a_1 + 2a_2 t_0 + 3a_3 t_0^2 + 4a_4 t_0^3 + 5a_5 t_0^4$$

$$v(t_f) = a_1 + 2a_2 t_f + 3a_3 t_f^2 + 4a_4 t_f^3 + 5a_5 t_f^4$$

$$a(t_0) = 2a_2 + 6a_3 t_0 + 12a_4 t_0^2 + 20a_5 t_0^3$$

$$a(t_f) = 2a_2 + 6a_3 t_f + 12a_4 t_f^2 + 20a_5 t_f^3$$

Insert the equations into augmented matrix form

$$\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & q(t_0) \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 & q(t_f) \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 & v(t_0) \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 & v(t_f) \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 & a(t_0) \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 & a(t_f) \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ 1 \end{bmatrix}$$

Use Gauss Jordan elimination to the augmented matrix to make it reduced row echelon form

$$rref(\begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 & t_0^4 & t_0^5 & q(t_0) \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 & q(t_f) \\ 0 & 1 & 2t_0 & 3t_0^2 & 4t_0^3 & 5t_0^4 & v(t_0) \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 & v(t_f) \\ 0 & 0 & 2 & 6t_0 & 12t_0^2 & 20t_0^3 & a(t_0) \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 & a(t_f) \end{bmatrix}) \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ 1 \end{bmatrix}$$

Multiply the reduced augmented matrix by the coefficient vector

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & k_0 \\ 0 & 1 & 0 & 0 & 0 & 0 & k_1 \\ 0 & 0 & 1 & 0 & 0 & 0 & k_2 \\ 0 & 0 & 0 & 1 & 0 & 0 & k_3 \\ 0 & 0 & 0 & 0 & 1 & 0 & k_4 \\ 0 & 0 & 0 & 0 & 0 & 1 & k_5 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ 1 \end{bmatrix}$$

a_n is a solely a function of k_n

$$\begin{bmatrix} a_0 & 0 & 0 & 0 & 0 & 0 & k_0 \\ 0 & a_1 & 0 & 0 & 0 & 0 & k_1 \\ 0 & 0 & a_2 & 0 & 0 & 0 & k_2 \\ 0 & 0 & 0 & a_3 & 0 & 0 & k_3 \\ 0 & 0 & 0 & 0 & a_4 & 0 & k_4 \\ 0 & 0 & 0 & 0 & 0 & a_5 & k_5 \end{bmatrix}$$

$\therefore \vec{a} = \vec{k}$

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix} = \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \end{bmatrix}$$

This trajectory planning is implemented in task space by solving for a cubic Bezier curve with control points used to avoid obstacles in the same plane. During motion, the current time vector is multiplied by the coefficient vector to determine the current parametric parameter, which is then input into the Bezier curve to find the end effector in task space. This position is then run through inverse kinematics to determine the joint angles to write the arm to to reach this point in space. This process is repeated to produce the trajectory.

$$\begin{bmatrix} t \end{bmatrix} = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 & a_4 & a_5 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \\ t^4 \\ t^5 \end{bmatrix}$$

$$\begin{bmatrix} p_{eex} \\ p_{eey} \\ p_{eez} \end{bmatrix} = (1-t)^3 \vec{p_0} + 3t(1-t)^2 \vec{p_1} + 3t^2(1-t)\vec{p_2} + t^3 \vec{p_1}$$

This quintic polynomial planning produces a smooth and continuous trajectory that starts and stops with zero velocity and acceleration. This is then input into the parametric Bezier equation to find the current target point that is then ran through inverse kinematics to find the current joint angles. Beziar trajectory was tested and implemented to allow the robot to allow the robot to grab objects without bumping into other objects. The quintic trajectory was used to move along this trajectory.

F. Camera Vision

To allow the robot to interact with its environment, a camera was used to allow the robot to determine the location of objects of interest. A checkerboard was placed in front of the robot to facilitate this. To determine the location of an object in the Task Space, three transformations must be computed, from camera space to pixel space (intrinsic camera matrix), from pixel to checkerboard space, and from checkerboard space to task space.

The intrinsic calibration and pixel-to-checkerboard space matrices are computed automatically by MATLAB. Many pictures of the checkerboard are taken from different vantage points. The checkerboard provides a large number of known features and a known origin. This allows the focal length, principal point, and distortion coefficients to be computed. The transformation matrix between the pixel space and the checkerboard board is computed by recognizing the origin of the checkerboard, and computing the transformation matrix from pixel space to checkerboard space. A pinhole approximation is used to compute this transformation (See Figure 3).

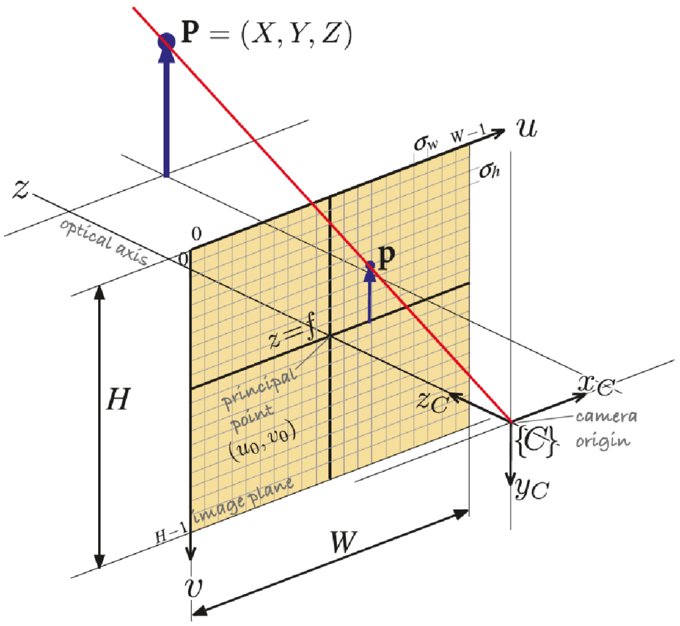

Fig. 3. Pinhole Camera Approximation

The final transformation from checkerboard space to task space was computed by inspection. The transformation matrix consists of a rotation matrix that was computed by inspection, and a translation vector that was computed by measuring offsets.

$$T_0^{Check} = \begin{bmatrix} x_{Checker} \cdot x_0 & y_{Checker} \cdot x_0 & z_{Checker} \cdot x_0 & \Delta x \\ x_{Checker} \cdot y_0 & y_{Checker} \cdot y_0 & z_{Checker} \cdot y_0 & \Delta y \\ x_{Checker} \cdot z_0 & y_{Checker} \cdot z_0 & z_{Checker} \cdot z_0 & \Delta z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The coordinate frames were assigned both to the checkerboard as the base of the robot (See Figure 4) and the transformation matrix, $T_0^{Checker}$ computed.

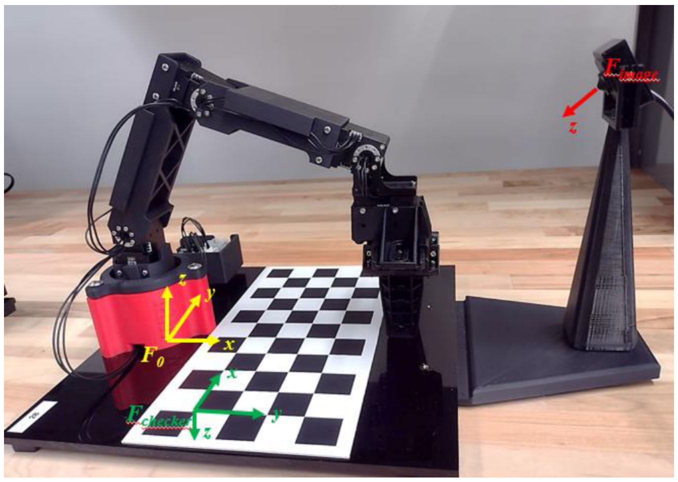

Fig. 4. Checkerboard and Base Frames

$$T_0^{Checker} = \begin{bmatrix} 0 & 1 & 0 & 99 \\ 1 & 0 & 0 & -114 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The position in the task space can then be computed by multiplying the transformation matrix by the position in checkerboard space.

$$\begin{bmatrix} x_{task} \\ y_{task} \\ z_{task} \\ 1 \end{bmatrix} = T_0^{Checker} \begin{bmatrix} x_{check} \\ y_{check} \\ z_{check} \\ 1 \end{bmatrix}$$

To determine the centroids of the objects in pixel space and determine their color. The following imaging processing flow was developed: mask everything except for checkerboard $\rightarrow$ convert image to HSV space $\rightarrow$ generate threshold images for each color (red, orange, yellow, green, blue) $\rightarrow$ detect edges $\rightarrow$ fill closed shapes $\rightarrow$ erode images $\rightarrow$ determine the centroid of any connected areas with an area of larger then 100 pixels.

The area outside of the checkerboard was masked by determining the locations of the corners of the checkerboard, adding the distance of one checker (to ensure the outer row of checkers was included), and creating a binary mask to set all pixels outside of the checkerboard to black. This prevents object recognition in areas the robot cannot reach (See Figure 5).

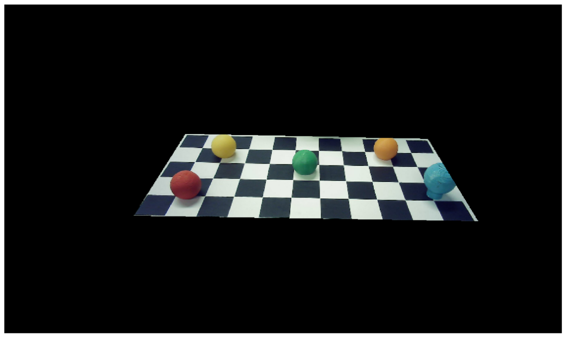

Fig. 5. Image with Masking for Area Outside of Workspace

The HSV (hue, saturation, value) colorspace was chosen because it was experimentally determined to provide the best color distinction between our red, orange, yellow, green, and blue objects.

Threshold images were created for each color, with all objects of the color shown in white, and the rest of the image in black. The edges were detected using a log algorithm as it was determined to be more sensitive then a Sobel filter (See Figure 6).

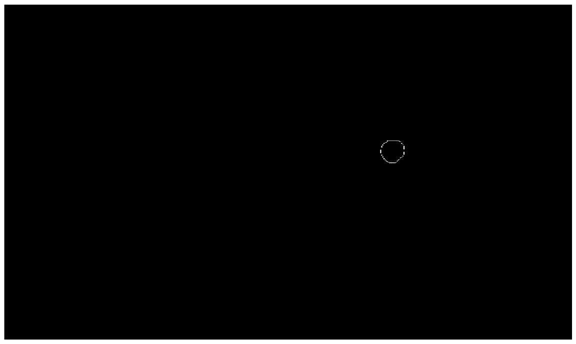

Fig. 6. Orange Image Edge Detection

The closed shapes generated by the edge detection were then filled (See Figure 7).

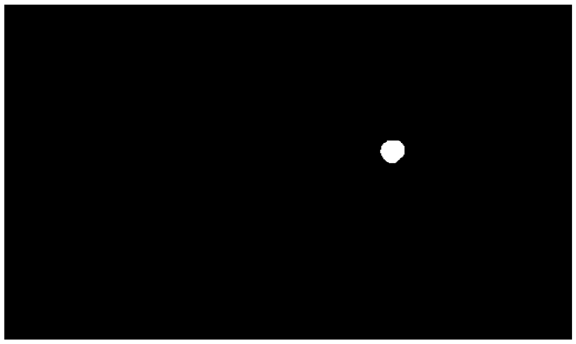

Fig. 7. Filled Shapes Images

The areas were eroded to decrease the area of small artifacts. The remaining areas were filtered by area to ensure artifacts were filtered out. The centroids of these areas in pixel space was then returned (See Figure 8).

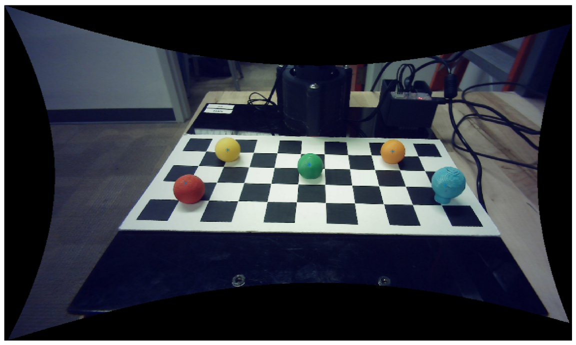

Fig. 8. Centroids in Pixel Space

This pixel space centroids can then be converted into task space using the transformations described above. However, these task space positions must be adjusted to ensure correct 3D coordinates. When mapping a 2D image into 3D, the position of the ball is determined to be farther away from the camera then it actually is, because the object has a z-height (See Figure 9).

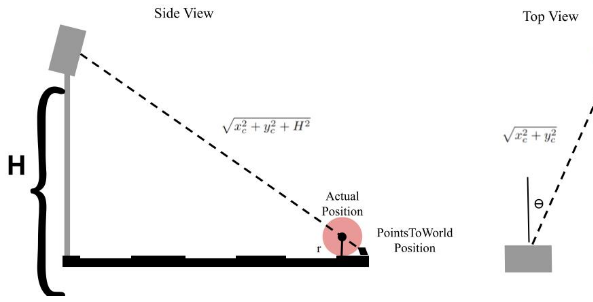

Fig. 9. Task Space Compensation

$$\text{offset} = \frac{r}{H}\sqrt{x_c^2+y_c^2}$$
$$x_{off} = \text{offset} * \cos(\theta)$$
$$y_{off} = \text{offset} * \sin(\theta)$$

G. Architecture

To allow for support for future expansion and maintainability of code base, and object oriented framework was followed. The following classes were created to handle various functions of the robot:

1) VirtualRobot: a Parent Class of Robot, allowing for trajectories to be run in a simulation (liveplot) before running on robot hardware

2) Robot: handles commands that require the physical robot hardware to run, including joint angle reading and writing functions, estop functions, and runnable trajectory functions
3) KinematicFunctions: handles forward and inverse position and velocity kinematic functions, that do not require the robot to run
4) TrajPlanner: handles generating coefficients for trajectory generation
5) Camera: handles functions that require the camera to run, including calibration scripts, transformation scripts, and image processing

H. Integration

The to integrate the aforementioned functionalities into an object sorting robot, a state machine was constructed (See Figure 10).

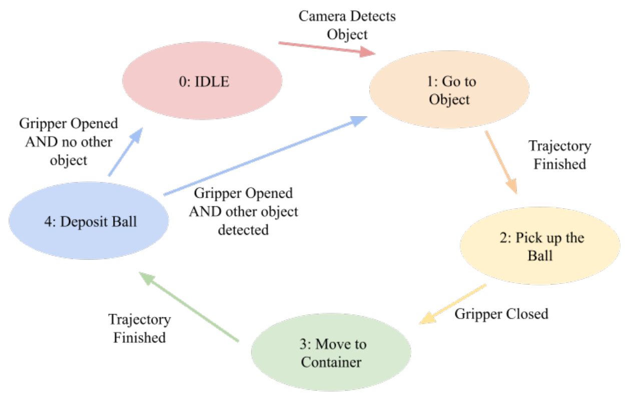

Fig. 10. State Machine Diagram

This implementation consisted of five states, while checking the camera every loop to keep the list of balls updated:

1) Idle: When robot does not see any objects, returns to home configuration
2) Go to Object: Generate a Bezier trajectory between the current location and the object position, and follow trajectory
3) Pick up Ball: Close gripper around object
4) Move to Container: Based on color of ball, generate Bezier trajectory between current position and respective container, and follow trajectory
5) Deposit Ball: Open Gripper

To implement active object tracking, a script was written to determine the task space positions of the object in the task space. A z-axis offset was added to this position and it was run through inverse kinematics to determine the joint angles to reach the point over the ball. The robot was sent to interpolate to this point. If the ball was observed to remain stationary, then the robot was instructed to approach and grab the ball.

Other objects were tested for collection by training the robot on other colors of blue, and presenting the robot with a dark blue marker cap. The robot recognized and sorted the market cap with the other blue objects.

Discussion

I. Forward Kinematics

The forward kinematics of the robot were developed and refined to determine the task space position of the end effector of the robot given the joint angles of the robot. The Denavit-Hartenburg convention was used to determine the intermediate transformation matricies. These were precomputed and transformed into MATLAB functions to improve speed of use. The speed of the forward kinematic functions is of great importance because the forward kinematics are called on every iteration of inverse kinematics (forward kinematics could be called up to 50 times per inverse kinematic call).

J. Differential Kinematics

The differential kinematics of the robot were determined by computing the Jacobian symbolically, then storing it as a MATLAB function. The Jacobian allows for determination of joint space velocities required to produce a specified end effector velocity, and vice versa. This function was precomputed on start up and stored as a fast MATLAB function, because it is also called on every iteration of inverse kinematics.

K. Inverse Kinematics

The inverse kinematics of the robot was computed numerically for decreased complexity from an algebraic approach. The Newton-Raphson root finding method was used. It is an iterative approach that was found to converge upon joint angles within fifty iterations. This method also does not require an α (angle of the end effector from horizontal) to be specified. This allows for reduced complexity when path planning this is because for most movements alpha can be a range of values for the trajectory to execute correctly however when alpha is specified it can lead to crashes in the algebraic inverse kinematics since the robot cant reach that point at that alpha, this is primarily prevalent for the extremes at the edge of the workspace and close to the origin. The solution was restricted to an elbow up configuration to allow the robot to reach more positions.

L. Trajectory Generation

To generate trajectories between objects, a quintic trajectory is generated for a quintic Bezier curve. The quintic trajectory allows the robot to move smoothly, with defined start and end velocities and accelerations along the path. The Bezier curve was defined to allow the robot to move from object to object without requiring setpoints, or knocking into objects. The control points are set to vertical extensions of the target point this ensures that the robot takes a path that goes up goes over and goes down to the target.

M. Camera Vision

To determine the location of objects, the camera was employed to recognize colored objects. The camera was trained on red, orange, yellow, green, and blue colored objects in HSV space for improved differentiation. Black and while objects could not be used because the checkerboard background is black and white, causing the checkerboard to be recognized as an object. The camera space position of the objects are converted to pixel space using the intrinsic parameters of the camera. Using the pixel space to checkerboard transformation, the position of the object in checkerboard space was computed. The position of the object in task space was then computed using the transformation matrix from checkerboard space to task space. This point was then adjusted using the known radius of the object.

The centroids of the objects were determined using color segmentation, and an image processing flow that was determined through experimentation. This flow was settled on as it provides reliable object detection in a variety of lighting conditions and provides easy color distinction. It also provided consistently accurate centroid detection.

N. Architecture

The code base was organized following object oriented principals to allow for future expansion. A parent class of Robot, VirtualRobot was created to allow the robots trajectories to be run in simulation before being run on hardware. This allows for easy debugging without the risk of damaging the hardware.

O. Integration

The previous functionality was compiled into a state machine to allow the robot to cycle between idle, moving to objects, picking up objects, moving to correct container, and dropping objects. This allows the robot to sort many balls one after another, without having to return home, minimizing the path length of the robot. The robot sorts balls in rainbow order, starting with red. Each of the five trained colors have their own containers placed outside of the checkerboard. By placing the containers outside of the checkerboard, it is masked out and already sorted balls are excluded from future movements. Live object tracking and additional object picking up were implemented. The camera was trained to follow above a ball (to prevent it from blocking the camera) until the ball remains still, then pick it up. The camera was trained on an additional hue of blue to allow the robot to pick up marker caps from the field.

Conclusion

To implement a object sorting system using a four Degree of Freedom articulated arm, various control techniques were employed. Forward and inverse position kinematics were developed to determine the joint angles from a task space pose and vise versa. This allows the robot to interact with its environment in task space. Forward and inverse velocity kinematics were developed to allow for determination of joint space velocities required to produce a task space velocity and vise versa. This allows for end effector task space speed control. Trajectory generation was implemented to allow the robot to follow a smooth Bezier curve from object to object. It also prevents strain on the motors by controlling the velocity and acceleration on the motors. To allow the robot to interact with its environment Camera Vision was implemented. This allows the robot to determine the color and location of objects in its surroundings so it can sort them. These techniques were integrated to create a system capable of sorting five distinct colors of objects.

References

[1] P. I. Corke, ”A Simple and Systematic Approach to Assigning Denavit–Hartenberg Parameters,” in IEEE Transactions on Robotics, vol. 23, no. 3, pp. 590-594, June 2007, doi: 10.1109/TRO.2007.896765. keywords: Robot kinematics;DH-HEMTs;Service robots;Manipulator dynamics;Motion planning;Java;Code standards;Algebra;Computational geometry;Jacobian matrices;Denavit–Hartenberg (DH);kinematics,

[2] S. Lee, J. Lee, J. Bang and J. Lee, ”7 DOF Manipulator Construction and Inverse Kinematics Calculation and Analysis using Newton-Raphson Method,” 2021 18th International Conference on Ubiquitous Robots (UR), Gangneung, Korea (South), 2021, pp. 235-238, doi: 10.1109/UR52253.2021.9494699. keywords: Kinematics;Manipulators;Hardware;Newton method,

Appendix

Code Base:
https://github.com/RBE3001-A24/RBE3001_A24_Team_5

Video:
https://www.youtube.com/watch?v=TpuRt1VqW_Y

Section	Contributor
Report	Sarah Listzwan and Zachary Serocki
Video	Alexander Kraemling
Programming	Zachary Serocki, Sarah Listzwan and Alexander Kraemling

Fig. 11. Contributions Table

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{9016}{250} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 12. Symbolic T_0^1

$$\begin{bmatrix} c_{\theta_1} & 0 & -s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & c_{\theta_1} & 0 \\ 0 & -1 & 0 & \frac{9016}{250} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 13. Symbolic T_0^2

$$\begin{bmatrix} c_{\theta_1}c_{\theta_2} & -c_{\theta_1}s_{\theta_2} & -s_{\theta_1} & 130.23c_{\theta_1}c_{\theta_2} \\ s_{\theta_1}c_{\theta_2} & -s_{\theta_1}s_{\theta_2} & c_{\theta_1} & 130.23s_{\theta_1}c_{\theta_2} \\ -s_{\theta_2} & -c_{\theta_2} & 0 & \frac{48163}{500} - 130.23c_{\theta_2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 14. Symbolic T_0^3

$$\begin{bmatrix} c_{\theta_4}(c_{\theta_1}c_{\theta_2}c_{\theta_3} - c_{\theta_1}s_{\theta_2}s_{\theta_3}) - s_{\theta_4}(c_{\theta_1}c_{\theta_2}s_{\theta_3} + c_{\theta_1}c_{\theta_3}s_{\theta_2}) & -c_{\theta_4}(c_{\theta_1}c_{\theta_2}s_{\theta_3} + c_{\theta_1}c_{\theta_3}s_{\theta_2}) - s_{\theta_4}(c_{\theta_1}c_{\theta_2}c_{\theta_3} - c_{\theta_1}s_{\theta_2}s_{\theta_3}) & -s_{\theta_1} & \frac{667c_{\theta_4}(c_{\theta_1}c_{\theta_2}c_{\theta_3} - c_{\theta_1}s_{\theta_2}s_{\theta_3})}{5} - \frac{667s_{\theta_4}(c_{\theta_1}c_{\theta_2}s_{\theta_3} + c_{\theta_1}c_{\theta_3}s_{\theta_2})}{5} + 8\sqrt{265}c_{\theta_1}c_{\theta_2} + 124c_{\theta_1}c_{\theta_2}c_{\theta_3} - 124c_{\theta_1}s_{\theta_2}s_{\theta_3} \\ c_{\theta_4}(s_{\theta_1}c_{\theta_2}c_{\theta_3} - s_{\theta_1}s_{\theta_2}s_{\theta_3}) - s_{\theta_4}(s_{\theta_1}c_{\theta_2}s_{\theta_3} + s_{\theta_1}c_{\theta_3}s_{\theta_2}) & -c_{\theta_4}(s_{\theta_1}c_{\theta_2}s_{\theta_3} + s_{\theta_1}c_{\theta_3}s_{\theta_2}) - s_{\theta_4}(s_{\theta_1}c_{\theta_2}c_{\theta_3} - s_{\theta_1}s_{\theta_2}s_{\theta_3}) & c_{\theta_1} & \frac{667c_{\theta_4}(s_{\theta_1}c_{\theta_2}c_{\theta_3} - s_{\theta_1}s_{\theta_2}s_{\theta_3})}{5} - \frac{667s_{\theta_4}(s_{\theta_1}c_{\theta_2}s_{\theta_3} + s_{\theta_1}c_{\theta_3}s_{\theta_2})}{5} + 8\sqrt{265}s_{\theta_1}c_{\theta_2} + 124s_{\theta_1}c_{\theta_2}c_{\theta_3} - 124s_{\theta_1}s_{\theta_2}s_{\theta_3} \\ s_{\theta_4}(s_{\theta_2}s_{\theta_3} - c_{\theta_2}c_{\theta_3}) - c_{\theta_4}(c_{\theta_2}s_{\theta_3} + c_{\theta_3}s_{\theta_2}) & c_{\theta_4}(s_{\theta_2}s_{\theta_3} - c_{\theta_2}c_{\theta_3}) + s_{\theta_4}(c_{\theta_2}s_{\theta_3} + c_{\theta_3}s_{\theta_2}) & 0 & \frac{667s_{\theta_4}(s_{\theta_2}s_{\theta_3} - c_{\theta_2}c_{\theta_3})}{5} - 124c_{\theta_3}s_{\theta_2} - \frac{667c_{\theta_4}(c_{\theta_2}s_{\theta_3} + c_{\theta_3}s_{\theta_2})}{5} - 124c_{\theta_2}s_{\theta_3} - 8\sqrt{265}s_{\theta_2} + \frac{48163}{500} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 15. Symbolic T_0^4

$$\begin{bmatrix} \begin{matrix} c_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a}c_{\theta_1} - s_{\theta_2-a}s_{\theta_3+a}c_{\theta_1}) - s_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a}c_{\theta_1} + c_{\theta_3+a}s_{\theta_2-a}c_{\theta_1}) \\ +bc_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a}c_{\theta_1} - s_{\theta_2-a}s_{\theta_3+a}c_{\theta_1}) + ec_{\theta_2-a}c_{\theta_3+a}c_{\theta_1} - es_{\theta_2-a}s_{\theta_3+a}c_{\theta_1} \end{matrix} & -s_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}) - c_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}) & -s_{\theta_1} & dc_{\theta_2-a}c_{\theta_1} - bs_{\theta_4}(c_{\theta_2-c}s_{\theta_3+a}c_{\theta_1} + c_{\theta_3+a}s_{\theta_2-a}c_{\theta_1}) \\ \begin{matrix} c_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a}s_{\theta_1} - s_{\theta_2-a}s_{\theta_3+a}s_{\theta_1}) - s_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a}s_{\theta_1} + c_{\theta_3+a}s_{\theta_2-a}s_{\theta_1}) \\ +bc_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a}s_{\theta_1} - s_{\theta_2-a}s_{\theta_3+a}s_{\theta_1}) + ec_{\theta_2-a}c_{\theta_3+a}s_{\theta_1} - es_{\theta_2-a}s_{\theta_3+a}s_{\theta_1} \end{matrix} & -c_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a}s_{\theta_1} + c_{\theta_3+a}s_{\theta_2-a}s_{\theta_1}) - s_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a}s_{\theta_1} - s_{\theta_2-a}s_{\theta_3+a}s_{\theta_1}) & c_{\theta_1} & dc_{\theta_2-a}s_{\theta_1} - bs_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a}s_{\theta_1} + c_{\theta_3+a}s_{\theta_2-a}s_{\theta_1}) \\ \begin{matrix} -s_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}) - c_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}) \\ -ec_{\theta_3+a}s_{\theta_2-a} - bc_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}) - ds_{\theta_2-a} \end{matrix} & s_{\theta_4}(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}) - c_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}) & 0 & f - bs_{\theta_4}(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}) - ec_{\theta_2-a}s_{\theta_3+a} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Fig. 16. Symbolic T_0^{EE}

$$\begin{bmatrix} -c\left[c_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right) - s_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + s_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right)\right] + es_{\theta_1}c_{\theta_2-a} + bs_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - bs_{\theta_1}s_{\theta_2-a}s_{\theta_3+a} & -c\left[c_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - c_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] + ec_{\theta_1}s_{\theta_2-a} + bc_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + bc_{\theta_1}c_{\theta_3+a}s_{\theta_2-a} & -c\left[c_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - c_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] + bc_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + bc_{\theta_1}c_{\theta_3+a}s_{\theta_2-a} & -\frac{d}{5}\left[c_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - c_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] \\ c\left[c_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - c_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right) - s_{\theta_4}\left(c_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right)\right] + ec_{\theta_1}c_{\theta_2-a} + bc_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - bc_{\theta_1}s_{\theta_2-a}s_{\theta_3+a} & -c\left[c_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + s_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] + es_{\theta_1}s_{\theta_2-a} + bs_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + bs_{\theta_1}c_{\theta_3+a}s_{\theta_2-a} & -c\left[c_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + s_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] + bs_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + bs_{\theta_1}c_{\theta_3+a}s_{\theta_2-a} & -\frac{d}{5}\left[c_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}s_{\theta_3+a} + s_{\theta_1}c_{\theta_3+a}s_{\theta_2-a}\right) + s_{\theta_4}\left(s_{\theta_1}c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_1}s_{\theta_2-a}s_{\theta_3+a}\right)\right] \\ 0 & c\left[c_{\theta_4}\left(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}\right) - s_{\theta_4}\left(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}\right)\right] + bc_{\theta_2-a}c_{\theta_3+a} - bs_{\theta_2-a}s_{\theta_3+a} + ec_{\theta_2-a} & c\left[s_{\theta_4}\left(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}\right) - c_{\theta_4}\left(c_{\theta_2-a}c_{\theta_3+a} - s_{\theta_2-a}s_{\theta_3+a}\right)\right] - bc_{\theta_2-a}c_{\theta_3+a} + bs_{\theta_2-a}s_{\theta_3+a} & \frac{d}{5}\left[c_{\theta_4}\left(s_{\theta_2-a}s_{\theta_3+a} - c_{\theta_2-a}c_{\theta_3+a}\right) + s_{\theta_4}\left(c_{\theta_2-a}s_{\theta_3+a} + c_{\theta_3+a}s_{\theta_2-a}\right)\right] \\ 0 & -s_{\theta_1} & -s_{\theta_1} & -s_{\theta_1} \\ 0 & c_{\theta_1} & c_{\theta_1} & c_{\theta_1} \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Fig. 17. Symbolic Jacobian