RBE 3001 Final Project; Robotic Pick and Place System

Shivangi Sirsiwal, Yuancen Pu, and Matheos Simantirakis

***Abstract*— Previous lab assignments have built knowledge of forward and inverse kinematics, trajectory generation, and velocity kinematics. In the final project, these skills will be integrated with computer vision, enabling the robot to identify and manipulate objects to sort colored balls on a checkerboard workspace.**

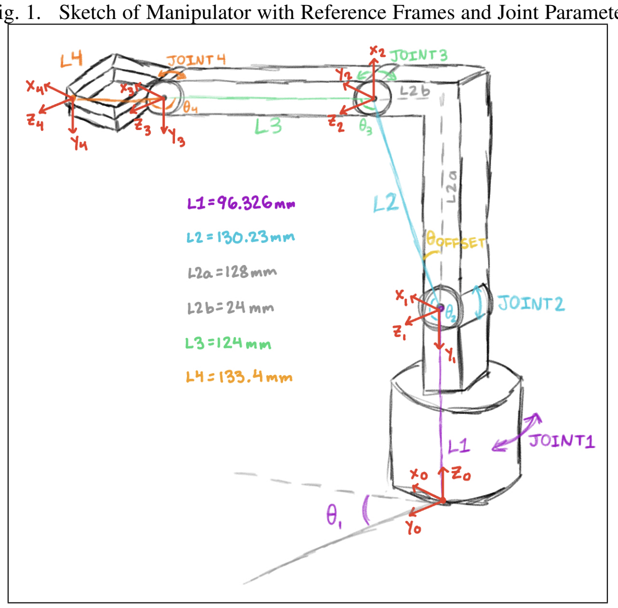
I. INTRODUCTION

The objective of this project is to create a robotic pick-and-place ball sorting system. Doing so will utilize forward and inverse position and velocity kinematics, trajectory planning, and computer vision to direct the Open Manipulator-X arm to detect spheres of different colors on the checkerboard workspace and sort them according to their color. This will make use of the knowledge and experience gained in previous labs while also homogenizing the functions created in these labs to work together toward this final goal.

II. PROPERTIES OF THE OPEN MANIPULATOR-X

A. Derivation of Robot's Forward Position Kinematics

Fig. 1. Sketch of Manipulator with Reference Frames and Joint Parameters



In order to derive the forward position kinematics transformation, it is useful to first start by drawing a sketch of the arm and assigning frames of reference to each joint following DH convention as seen in Figure 1.

Upon completing this, we were able to utilize our knowledge of the robot's dimensions and the orientations of the reference frames to identify the DH parameters (Table I) for each intermediate transform. We used these parameters

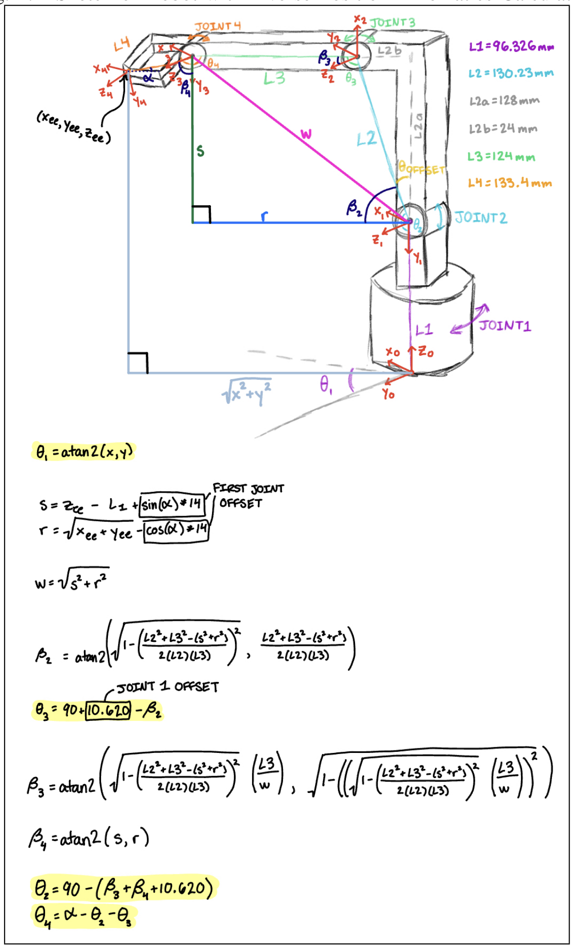
TABLE I. DH Parameters of the Open Manipulator-X Arm

	θ	d	a	α
T_0^1	θ_1^*	96.326	0	-90
T_1^2	θ_2^* - 79.3803	0	130.21	0
T_2^3	θ_3^* - 79.3803	0	124	0
T_3^4	θ_4^*	0	133.4	0

to find the intermediate transformation matrices (these can be found in section VIII, the appendix, in (Figure 13)) which we multiplied together to find the full base to end effector forward position kinematics transformation (the full expanded form of the symbolic forward position kinematics matrix can also be found in section VIII, Table II).

B. Derivation of Robot's Inverse Position Kinematics

Fig. 2. Sketch of Robot with Inverse Position Kinematics Calculations


JOINT4
L4
X_4
Z_4
Y_4
α
β_4
X_3
Y_3
θ_4
(xee, yee, zee)
X_2
JOINT3
Y_2
Z_2
β_3
θ_3
L3
L2b
W
S
L2
L2a
θ_{OFFSET}
β_2
r
X_1
Z_1
Y_1
θ_2
JOINT2
L1
JOINT1
L1=96.326mm
L2=130.23mm
L2a=128mm
L2b=24mm
L3=124mm
L4=133.4mm
X_0
Z_0
Y_0
$\sqrt{x^2+y^2}$
θ_1
$\theta_1 = atan2(x,y)$
$s = z_{ee} - L_1 + \boxed{\sin(\alpha)*14}$
$r = \sqrt{x_{ee} + y_{ee}} - \boxed{\cos(\alpha)*14}$
FIRST JOINT OFFSET
$w = \sqrt{s^2+r^2}$
$\beta_2 = atan2\left(\sqrt{1-\left(\frac{L2^2+L3^2-(s^2+r^2)}{2(L2)(L3)}\right)^2}, \frac{L2^2+L3^2-(s^2+r^2)}{2(L2)(L3)}\right)$
JOINT 1 OFFSET
$\theta_3 = 90 + \boxed{10.620} - \beta_2$
$\beta_3 = atan2\left(\sqrt{1-\left(\frac{L2^2+L3^2-(s^2+r^2)}{2(L2)(L3)}\right)^2}\left(\frac{L3}{w}\right), \sqrt{1-\left(\left(\sqrt{1-\left(\frac{L2^2+L3^2-(s^2+r^2)}{2(L2)(L3)}\right)^2}\left(\frac{L3}{w}\right)\right)^2\right)}\right)$
$\beta_4 = atan2(s,r)$
$\theta_2 = 90 - (\beta_3 + \beta_4 + 10.620)$
$\theta_4 = \alpha - \theta_2 - \theta_3$

In order to derive the inverse position kinematics of the robot we started by drawing a sketch of our robot with the joints and reference frames labeled. We then sketched various triangles relating the joint angles to the position of the end effector and used our knowledge of the robot's dimensions and trigonometric properties to derive the angles in relation to the end effector's position and orientation. This process can be seen in Figure 2.

C. Derivation of Robot's Forward Velocity Kinematics

Fig. 3. Calculations for the Forward Velocity Kinematics Jacobian

$$P_x = \frac{1302(\cos\theta_1^*)}{100} + 124(\cos(\theta_1^*))\cos(\theta_2^* - 79.3803) + \frac{667\cos(\theta_1^*)\cos(\theta_2^* - 79.3803)\cos(\theta_3^* - 79.3803)}{5} - \frac{667(\cos(\theta_1^*))\sin(\theta_2^* - 79.3803)\sin(\theta_3^* - 79.3803)}{5}$$

$$P_y = \frac{1302\sin(\theta_1^*)}{100} + 124\cos(\theta_2^* - 79.3803)\sin(\theta_1^*) + \frac{667\cos(\theta_2^* - 79.3803)\cos(\theta_3^* - 79.3803)\sin(\theta_1^*)}{5} - \frac{667\sin(\theta_1^*)\sin(\theta_2^* - 79.3803)\sin(\theta_3^* - 79.3803)}{5}$$

$$P_z = \frac{48163}{500} - \frac{667\cos(\theta_2^* - 79.3803)\sin(\theta_3^* - 79.3803)}{5} - \frac{667\cos(\theta_3^* - 79.3803)\sin(\theta_2^* - 79.3803)}{5} - 124\sin(\theta_2^* - 79.3803)$$

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \frac{\partial p_x}{\partial \theta_3} & \frac{\partial p_x}{\partial \theta_4} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \frac{\partial p_y}{\partial \theta_3} & \frac{\partial p_y}{\partial \theta_4} \\ \frac{\partial p_z}{\partial \theta_1} & \frac{\partial p_z}{\partial \theta_2} & \frac{\partial p_z}{\partial \theta_3} & \frac{\partial p_z}{\partial \theta_4} \end{bmatrix}$$

The Jacobian was derived using the partial derivative approach (see Figure 3). The *px*, *py*, and *pz* equations were retrieved from the first three rows of the last column of the base-to-end-effector FK matrix. The partial derivatives of these equations were then taken with respect to θ_1, θ_2, θ_3, and θ_4. Derivatives of *px* are expressed in the first row of the Jacobian, derivatives of *py* in the second row, and *pz* in the third. Derivatives with respect to θ_1 are expressed in the first column, derivatives with respect to θ_2 in the second column, θ_3 in the third, and θ_4 in the fourth column. The bottom half of the Jacobian is comprised of the last column of the rotation matrix for each intermediate transformation as derived by the forward kinematics function, with the last column of the bottom half of the Jacobian being the last column of the base-to-end-effector rotation matrix. Upon creating the symbolic Jacobian (see Table III in section VIII, the appendix, for full expanded form of the forward velocity jacobian), it was implemented in a function with the symbolic variables swapped out for the input joint variables.

D. Derivation of Robot's Inverse Velocity Kinematics

The robot's inverse velocity kinematics were derived by taking the inverse of the Jacobian calculated in section II-C, which can then be multiplied by any set of joint velocities to derive the end effector velocity for those parameters.

$$\vec{v}_{ee} = J(q)^{-1} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \\ \dot{\theta}_4 \end{bmatrix}$$

III. METHODOLOGY

A. Camera Calibration and Setup

It is essential to first set up and calibrate the camera for this project, as this will allow for the identification of the spheres in the workspace as well as color differentiation for sorting. The calibration process consists of two main parts: intrinsic and extrinsic calibration respectively.

Fig. 4. Visualization of Extrinsic Parameters Used for Intrinsic Calibration

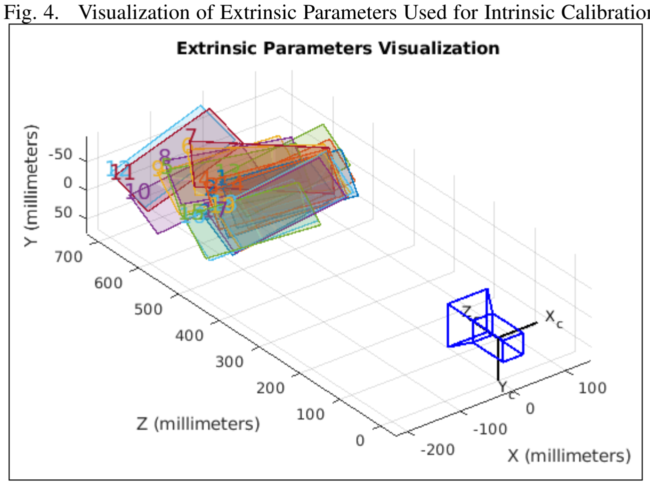
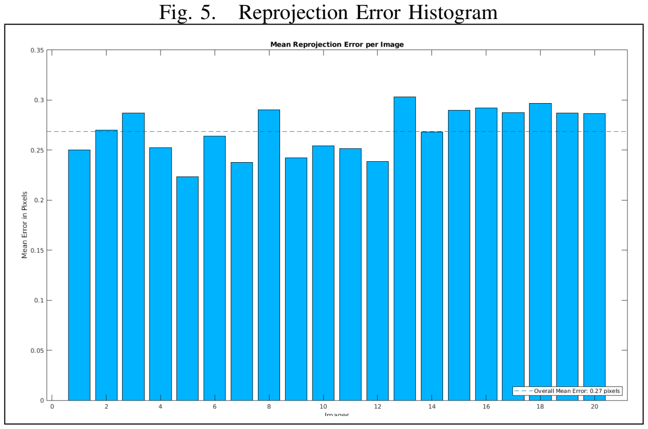


Fig. 5. Reprojection Error Histogram



1) Intrinsic Calibration: Intrinsic calibration was accomplished with the help of the camera calibration app available in MATLAB. One hundred photographs were taken of the workspace by the camera from various angles. Twenty images were selected from the hundred that displayed the axes in the correct orientations and displayed all twenty-seven intersection points in the correct locations in the workspace. The frames of these photos relative to the placement of the camera can be seen in Figure 4. The images were also tested to ensure that their reprojection errors were at an acceptable level, meaning that the difference between a point as seen in the image and the point as seen projected from the real world into the image is acceptably small. The reprojection error for the twenty images is represented in the histogram in Figure 5. The mean reprojection level is also shown on the histogram, it has a value of approximately 0.27 pixels. This is an acceptable error level.

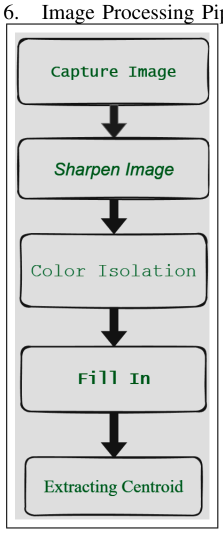
Given the dimensions of the checkerboard, the calibrator is able to fix the fish-eye distortion of the camera by ensuring the intersection points are arranged 25mm apart and oriented according to the axes. The camera calibrator uses this information intrinsically to understand how to convert the raw image captured by the camera into a pixel coordinate system that is an accurate reflection of the physical workspace.

2) Extrinsic Calibration: Upon the completion of intrinsic calibration, extrinsic calibration must be applied to interpret the pixel coordinate system into the dimensions of the base frame of the robot. This is achieved by multiplying the transformation matrix generated by intrinsic calibration by the transformation matrix to go from the workspace frame to the robot base frame. This completes the transformation from the x,y frame of the image to the x,y,z base frame of the robot. This transformation is essential to the robot understanding and being capable of acting on the images captured by the camera.

B. Image Processing

Upon obtaining the image and properly transforming it through camera calibration, it is then necessary to process the image to extract the information needed for the purposes of this project.

Fig. 6. Image Processing Pipeline

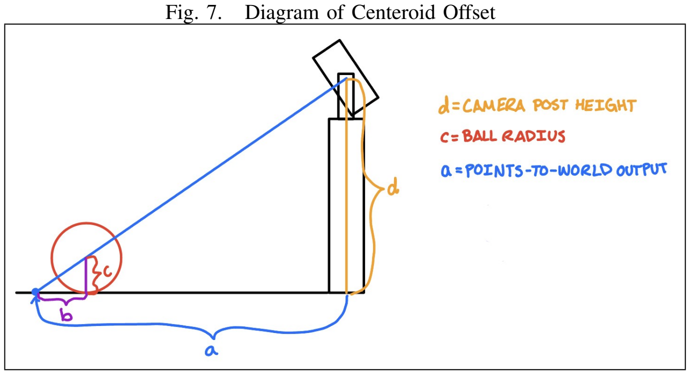


This is done by first masking everything in the image save for the checkerboard. This removes any possible interference from other objects not in the workspace. Once the workspace is isolated, color identification is implemented using the HSV color space. This works by taking hue, saturation, and vibrancy ranges for a color and analyzing the image matrix for pixels that fall within those three ranges. That pixel is then identified to be that color and any pixels that are not within the range are filtered out.

HSV was determined to be the best color space for the project as it provides greater precision in identification. HSV better fits the conditions as it accounts for highlights and shadows through the vibrancy parameters, as well as varying lighting conditions with the saturation parameters.

Once the desired color has been identified in the image, everything else is masked out. MATLAB's built in find circle function then identifies the areas in the image that are circular with a radius close to the desired radius and fills in any gaps that may have resulted from the color filtering. This same function provides the centroids of the identified circles.

Fig. 7. Diagram of Centeroid Offset



This image processing procedure is represented visually in Figure 6.

The centroid of the circle in the image frame is slightly offset from the actual location of the ball in the workspace, though, as the ball is a three-dimensional object and the image frame only sees two dimensions. In order to resolve this offset, it is useful to first visualize the offset as in Figure 7. The right triangle formed by the camera post, the camera's view line, and the points-to-world output and the right triangle formed by the camera view line, the radius of the ball, and the offset are nested right triangles. This means that the ratio of the camera post height to the ball's radius is equal to the ratio of the points-to-world output to the offset.

$$\frac{d}{c} = \frac{a}{b}$$

The camera post height (d), point-to-world output (a), and ball radius (c) are known values, thus the equation is easily solvable for b.

$$b = \frac{ac}{d}$$

This relationship is implemented in MATLAB to determine the true (x,y,z) coordinates of the centroid of the ball in the 3D workspace.

C. Robot Trajectory Planning and Sorting

Once the position of the ball in the task space is located, it is used as the target end effector position. In order to direct the robot to this position, the inverse position kinematics of the robot for this position is derived via MATLAB and quintic trajectory planning in the task space is implemented using this output. Quintic trajectory planning was determined to be the optimal trajectory planning style as it provides smooth and controlled motion. Once the end effector has arrived at the correct location, the gripper closes around the ball and the ball is lifted off of the workspace. The robot then takes the ball to a designated location (determined by the color of the ball) off of the workspace, and drops the ball there.

IV. RESULTS

Fig. 8. Camera Image with Field Isolation Mask

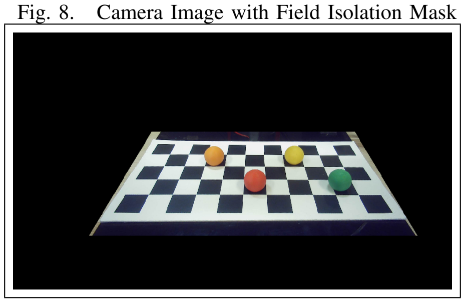


Fig. 9. Camera Image with Orange Isolation Mask

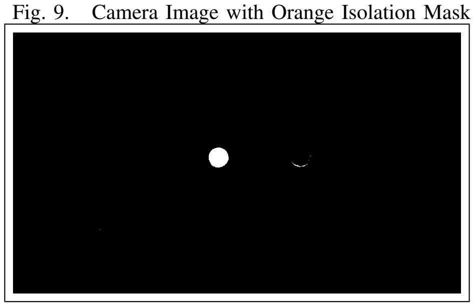


Fig. 10. Camera Image with Red Isolation Mask

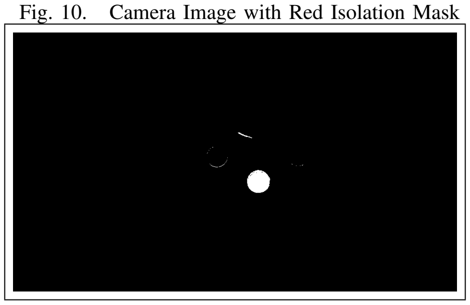


Fig. 11. Camera Image with Yellow Isolation Mask

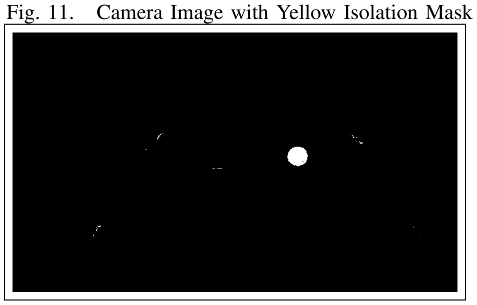
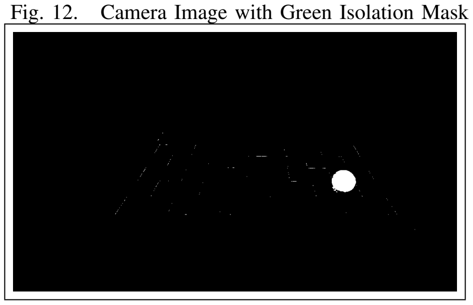


Fig. 12. Camera Image with Green Isolation Mask



V. DISCUSSION

The image outputs pictured in section IV give a visual representation of the accuracy of the ball detection system.

1) Field Isolation: Figure 8 shows the output of the workspace masking. There is a margin visible on all sides of the image. This margin is to account for the perspective of the camera; because the camera is placed at an angle, the balls placed along the edges of the workspace will spill slightly outside of the workspace when projected into the image frame. The margins in the workspace masking ensure this does not impact the ability of the system to detect the ball.

2) Orange Isolation: Figure 9 shows the output of detecting and isolating the color orange (as defined by set HSV parameters) in the image. Almost everything except for the ball has been completely filtered out. A bit of the bottom of yellow ball remains, this is likely because the lower vibrancy HSV parameters for orange and yellow have some overlap as both hues are similar. The bottom of the balls are generally darker and lower in vibrancy due to the placement of the lighting, so this could be causing the incorrect overlap. However the error is not substantial, and the shape of the error is not circular, so it will be ignored by the system.

3) Red Isolation: Figure 10 shows the output of detecting and isolating the color red (as defined by set HSV parameters) in the image. Again, the filtration is quite accurate, almost everything but the ball has been completely filtered out. There are some faint remnants of the orange and yellow balls still present in the image, which can be explained by how close to each other red, orange, and yellow are in hue. Again, the error is so small it is essentially negligible and will not be identified as a circle and interfere with the system. A streak of red is also detected toward the top center of the image, which can be explained by referencing Figure 8 and noting the small section of the red robot base visible in the image. This issue is easily corrected by ensuring the entirety of the robot base is covered, however, this is not necessary as the section of the base that is visible is not circular and thus will not interfere with the system.

4) Yellow Isolation: Figure 11 shows the output of detecting and isolating the color yellow (as defined by set HSV parameters) in the image. As expected, the filtration has removed almost everything but the ball. There are a few small areas of error that could be explained by noise in the system.

5) Green Isolation: Figure 12 shows the output of detecting and isolating the color green (as defined by set HSV parameters) in the image. The filtration was not as precise as expected, while the majority of the unwanted sections of the image have been removed, there are still some erroneous lines present throughout the image. In comparing this image to Figure 8 it is notable that almost all of the unwanted lines align with the edges of the black squares on the checkerboard. This indicates that the error might be caused by the vibrancy lower bound being too low. This would mean the error could be resolved by adjusting the vibrancy parameters to completely exclude the checkerboard, but this is unnecessary as the lines are not circular and thus will not interfere with the system.

In observing the behavior of the robot during system implementation, it is clear that the filtration, centroid location derivation, inverse kinematics, and trajectory planning are all working properly and homogeneously as the robot is able to consistently and accurately locate, lift, and sort the colored balls smoothly and with precision.

VI. CODE SYSTEMS

A. ***Robot.m***

Robot.m is the class that houses all the functions to move the robot.

1) Forward Position Kinematics(fk3001): In order to derive the end effector position for any set of joint values, a series of functions were created to implement the calculations in section II-A. First, the inputted joint angles are substituted into the symbolic DH table (Table I) to derive the DH table for the given joint configuration. These DH parameters are then put into a function that calls another function to derive the intermediate transformation matrices. These intermediate transforms are then multiplied together in the function that called them to find the base to end effector FK matrix.

2) Inverse Position Kinematics (ik3001): In order to derive the joint values needed to achieve an end effector position, a function was created to implement the calculations in section II-B. The function first calculates θ_1, then θ_3, then θ_2, and θ_4.

3) Trajectory Planning (run_trajectory): In order to create the trajectory polynomial, a function was created that accepts polynomial coefficients from the Traj_Planner class (VI-B), desired travel time, and the joint space to derive the quintic polynomial in the task space. It then sets the robot trajectory accordingly.

4) Jacobian Generation (jacobian): This function takes in the joint parameters and calculates the Jacobian for these values.

5) Velocity Kinematics (velocity_control): This function takes the initial and final positions as well as the desired travel time and utilizes the jacobian generated by the function from VI-A.4 to derive the joint velocities needed to achieve this.

B. ***Traj_Planner.m***

1) Quintic Coefficient Generation (quintic_traj): This function takes initial and final times, velocities, positions, and accelerations and generates the coefficients for the corresponding quintic polynomial.

C. ***objectDetect.m***

This class contains the functions used in color isolation.

1) HSV Parameters (objectDetect): This function stores the HSV color parameters information.

2) Workspace Masking (applyMask): This function takes an image as well as the coordinates of the workpace as found in the camera calibration and blocks out everything outside of these points.

3) Mask Generation (createMask): This function takes in an image as well as the desired HSV parameters and creates a mask according to those values.

4) Color Masking (findGreen) (findYellow) (findRed) (findOrange): These functions take an image and feed it along with the HSV parameters for their respective colors as stored in the *objectDetect* (VI-C.1) and feed them to the *createMask* (VI-C.3) function to generate color-isolated images.

5) Circle Creation (getBall): This function takes a masked image and a radius range and searches for circular objects in the image within the radius range. The function returns the center point of these objects as well as it's radius.

6) Converting to Robot Base Frame (switchToRobot): This function takes the camera position and the current end effector position and converts the image from the checkerboard frame to the robot base frame. It also returns the final end effector position which can be fed into the inverse kinematics function.

7) Adjust for Centroid Offset (ballXYonBoard): This function takes the x,y coordinates of the centroid in the image as found by the *getBall* function and adjusts the coordinates according to the offset as shown in Figure 7 to determine the real x,y,z coordinates of the centroid of the ball.

8) Implementing Color Sorting (getNextBallPosition): This function takes the points for the edges of the workspace in the image, the x,y position of the ball on the board in the image, as well as the minimum height the robot must lift the ball to remove it from the workspace, as well as the radius range of the ball. The function then implements all the other *objectDetect.m* functions to iterate through detecting each ball color and sorting them.

The function starts by detecting the orange balls and removing one of those, then finding the yellow balls and removing one of those, then red, then green until there are no balls left.

VII. CONCLUSIONS

This project presented an interesting challenge that implemented all the concepts explored in previous labs in this course. The knowledge gained from this experience is versatile and will be useful in future robotics endeavors.

VIII. APPENDIX

Fig. 13. Intermediate Transforms for Forward Position Kinematics

$$T_0^1 = \begin{bmatrix} \cos(\theta_1^*) & 0 & -\sin(\theta_1^*) & 0 \\ \sin(\theta_1^*) & 0 & \cos(\theta_1^*) & 0 \\ 0 & -1 & 0 & 96.326 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} \cos(\theta_2^* - 79.3803) & -\sin(\theta_2^* - 79.3803) & 0 & 130.21 \\ \sin(\theta_2^* - 79.3803) & \cos(\theta_2^* - 79.3803) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} \cos(\theta_3^* - 79.3803) & -\sin(\theta_3^* - 79.3803) & 0 & 124 \\ \sin(\theta_3^* - 79.3803) & \cos(\theta_3^* - 79.3803) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} \cos(\theta_4^*) & -\sin(\theta_4^*) & 0 & 133.4 \\ \sin(\theta_4^*) & \cos(\theta_4^*) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

TABLE II. Forward Position Kinematics Matrix

Matrix Index	Value
(1,1)	$\cos(\theta_4^*)(\cos(\theta_1^*)\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38) - \cos(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38)) - \sin(\theta_4^*)(\cos(\theta_1^*)\cos(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38) + \cos(\theta_1^*)\cos(\theta_3^* - 79.38)\sin(\theta_2^* - 79.38))$
(1,2)	$-\cos(\theta_4^*)(\cos(\theta_1^*)\cos(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38) + \cos(\theta_1^*)\cos(\theta_3^* - 79.38)\sin(\theta_2^* - 79.38)) - \sin(\theta_4^*)(\cos(\theta_1^*)\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38) - \cos(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))$
(1,3)	$-\sin(\theta_1^*)$
(1,4)	$(13021\cos(\theta_1^*))/100 + 124\cos(\theta_1^*)\cos(\theta_2^* - 79.38) + (667\cos(\theta_1^*)\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38))/5 - (667\cos(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))/5$
(2,1)	$\cos(\theta_4^*)(\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38)\sin(\theta_1^*) - \sin(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38)) - \sin(\theta_4^*)(\cos(\theta_2^* - 79.38)\sin(\theta_1^*)\sin(\theta_3^* - 79.38) + \cos(\theta_3^* - 79.38)\sin(\theta_1^*)\sin(\theta_2^* - 79.38))$
(2,2)	$-\cos(\theta_4^*)(\cos(\theta_2^* - 79.38)\sin(\theta_1^*)\sin(\theta_3^* - 79.38) + \cos(\theta_3^* - 79.38)\sin(\theta_1^*)\sin(\theta_2^* - 79.38)) - \sin(\theta_4^*)(\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38)\sin(\theta_1^*) - \sin(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))$
(2,3)	$\cos(\theta_1^*)$
(2,4)	$(13021\sin(\theta_1^*))/100 + 124\cos(\theta_2^* - 79.38)\sin(\theta_1^*) + (667\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38)\sin(\theta_1^*))/5 - (667\sin(\theta_1^*)\sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))/5$
(3,1)	$-\cos(\theta_4^*)(\cos(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38) + \cos(\theta_3^* - 79.38)\sin(\theta_2^* - 79.38)) - \sin(\theta_4^*)(\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38) - \sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))$
(3,2)	$\sin(\theta_4^*)(\cos(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38) + \cos(\theta_3^* - 79.38)\sin(\theta_2^* - 79.38)) - \cos(\theta_4^*)(\cos(\theta_2^* - 79.38)\cos(\theta_3^* - 79.38) - \sin(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))$
(3,3)	0
(3,4)	$48163/500 - (667\cos(\theta_2^* - 79.38)\sin(\theta_3^* - 79.38))/5 - (667\cos(\theta_3^* - 79.38)\sin(\theta_2^* - 79.38))/5 - 124\sin(\theta_2^* - 79.38)$
(4,1)	0
(4,2)	0
(4,3)	0
(4,4)	1

TABLE III. Forward Velocity Kinematics Jacobian

Matrix Index	Value
(1,1)	cos(θ_1^*) + sin(θ_3^*)(cos(θ_2^*)sin(θ_1^*)) + cos(θ_3^*)(sin(θ_1^*)sin(θ_2^*)) + L3sin(θ_3^*)(sin(θ_1^*)sin(θ_2^*)) - L2cos(θ_2^*)sin(θ_1^*) + L4cos(θ_4^*) (sin(θ_3^*)(sin(θ_1^*) sin(θ_2^*)) - cos(θ_3^*)(cos(θ_2^*)sin(θ_1^*) - L3 cos(θ_3^*)(cos(θ_2^*)sin(θ_1^*))
(1,2)	L2cos(θ_1^*)sin(θ_2^*) - L3sin(θ_3^*)(cos(θ_1^*)cos(θ_2^*) + L4sin(θ_4^*) (sin(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - cos(θ_3^*) (cos(θ_1^*)cos(θ_2^*)) - L4 cos(θ_4^*)(sin(θ_3^*)(cos(θ_1^*) cos(θ_2^*)) + cos(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - L3cos(θ_3^*)(cos(θ_1^*)sin(θ_2^*))
(1,3)	-L3sin(θ_3^*)(cos(θ_1^*)cos(θ_2^*) - L4sin(θ_4^*)(cos(θ_3^*) (cos(θ_1^*) cos(θ_2^*)) - sin(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - L3 cos(θ_3^*)(cos(θ_1^*) sin(θ_2^*) - L4cos(θ_4^*)(cos(θ_3^*) (cos(θ_1^*)sin(θ_2^*)) + sin(θ_3^*) (cos(θ_1^*) cos(θ_2^*)))
(1,4)	L4sin(θ_4^*)(sin(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - cos(θ_3^*)(cos(θ_1^*)cos(θ_2^*)) - L4cos(θ_4^*)(sin(θ_3^*)(cos(θ_1^*)cos(θ_2^*)) + cos(θ_3^*)(cos(θ_1^*) sin(θ_2^*))
(2,1)	L2cos(θ_1^*)cos(θ_2^*) - L4sin(θ_4^*)(sin(θ_3^*)(cos(θ_1^*) cos(θ_2^*))) + cos(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - L3sin(θ_3^*)(cos(θ_1^*)sin(θ_2^*)) - L4 cos(θ_4^*)(sin(θ_3^*)(cos(θ_1^*) sin(θ_2^*)) - cos(θ_3^*)(cos(θ_1^*)cos(θ_2^*))) + L3cos(θ_3^*)(cos(θ_1^*)*cos(θ_2^*))
(2,2)	-L2sin(θ_1^*)sin(θ_2^*) - L3sin(θ_3^*)(cos(θ_2^*)sin(θ_1^*)) + L4sin(θ_4^*) (sin(θ_3^*)(sin(θ_1^*) sin(θ_2^*) - cos(θ_3^*)(cos(θ_2^*)sin(θ_1^*))) - L4 cos(θ_4^*)(sin(θ_3^*)(cos(θ_2^*)sin(θ_1^*)) + cos(θ_3^*)(sin(θ_1^*)sin(θ_2^*))) - L3cos(θ_3^*)(sin(θ_1^*)sin(θ_2^*))
(2,3)	-L3sin(θ_3^*)(cos(θ_2^*)sin(θ_1^*) - L4sin(θ_4^*)(cos(θ_3^*)(cos(θ_2^*) sin(θ_1^*)) - sin(θ_3^*)(sin(θ_1^*)sin(θ_2^*))) - L3 cos(θ_3^*)(sin(θ_1^*) sin(θ_2^*)) - L4cos(θ_4^*)(cos(θ_3^*) (sin(θ_1^*)sin(θ_2^*)) + sin(θ_3^*) (cos(θ_2^*) sin(θ_1^*)))
(2,4)	L4sin(θ_4^*)(sin(θ_3^*)(sin(θ_1^*)sin(θ_2^*)) - cos(θ_3^*)(cos(θ_2^*)sin(θ_1^*))) - L4cos(θ_4^*)(sin(θ_3^*)(cos(θ_2^*)sin(θ_1^*)) + cos(θ_3^*) (sin(θ_1^*) sin(θ_2^*)))
(3,1)	0
(3,2)	-L4sin(θ_4^*)(-cos(θ_2^*)sin(θ_3^*) + cos(θ_3^*) sin(θ_2^*)) - L2cos(θ_2^*) + L4cos(θ_4^*)(-cos(θ_2^*)cos(θ_3^*) + sin(θ_2^*)sin(θ_3^*)) - L3cos(θ_2^*) cos(θ_3^*) + L3sin(θ_2^*)sin(θ_3^*)
(3,3)	L4cos(θ_4^*)cos(θ_3^*)cos(θ_2^*) + sin(θ_2^*)sin(θ_3^*) - L3cos(θ_3^*) cos(θ_2^*) - L4sin(θ_4^*)(sin(θ_3)cos(θ_2^*) - cos(θ_3^*)sin(θ_2^*)) - L3 - sin(θ_2^*)sin(θ_3^*)
(3,4)	L4cos(θ_4^*)(cos(θ_3^*)(-cos(θ_2^*)) + sin(θ_2^*) sin(θ_3^*)) - L4sin(θ_4^*) (sin(θ_3^*)(-cos(θ_2^*)) - cos(θ_3^*)sin(θ_2^*))
(4,1)	0
(4,2)	-sin(θ_1^*)
(4,3)	-sin(θ_1^*)
(4,4)	-sin(θ_1^*)
(5,1)	0
(5,2)	cos(θ_1^*)
(5,3)	cos(θ_1^*)
(5,4)	cos(θ_1^*)
(6,1)	1
(6,2)	0
(6,3)	0
(6,4)	0