RBE 3001: Robotic Pick and Place System

Sukriti Kushwaha, Istan Slamet, Tracy Yang

***Abstract*— This lab aims to combine concepts learned from previous labs, such as forward and inverse position kinematics, trajectory planning, and forward and inverse velocity analysis into a final project where a robotic sorting system is designed. Additionally, new concepts such as image processing, camera calibration, and object recognition were utilized to identify and sort balls in the robot's workspace according to their color.**

I. INTRODUCTION

The final project for RBE 3001 represents a culmination of the skills and topics learned from the first four labs completed in this course. Lab 1 allowed us to learn how to manipulate the movements of the 4-degree-of-freedom OpenManipulator robotic arm. The experiments primarily included analysis of each of the revolute joints in relation to one another while undergoing several different movements with and without interpolation. Lab 2 used forward kinematics calculations to determine the positions and orientations of the end effector from given input joint values. Joint values corresponding to each servo motor were sent into DH parameters and inserted into a forward kinematics matrix of its respective joint. All of the matrices were then multiplied together, solving for the position and orientation of the end effector. The purpose of Lab 3 was to formulate the inverse kinematics of the robot arm and perform trajectory planning in joint and task space using cubic and quintic polynomials through linear interpolation. Inverse kinematics was used to find the angles of each of the joints with respect to their home position, given the end-effector's pose. Using trajectory planning combined with inverse kinematics, we achieved smooth motion planning in the task space of the robot. In Lab 4 we implemented velocity kinematics to the robot arm by calculating the Jacobian matrix to solve for the linear and angular velocities in the task space. From there, we planned the trajectory using speed and direction commands and identified singular configurations through the determinant of the Jacobian matrix. All of the programming was done using Matlab and we used git to track the changes to our source code and manage branches and its related issues.

The requirements of the final project include determining the 3D position of an object with respect to the robot's reference frame through image processing techniques. From there, the robot must be able to sort objects based on their color using object recognition and follow a smooth trajectory to and from that location. The gripper must close to pick up the object and transport it to an arbitrary location. Once there, the gripper is opened to place the object. The robot must also be able to handle objects being placed on the checkerboard while it is in the middle of sorting. To complete this challenge, we implemented forward position kinematics, inverse position kinematics, and forward velocity kinematics to demonstrate the skills we have learned throughout the previous labs and successfully pick and place objects in the robot's task space. This project is important because the analysis of the kinematics of a robotic arm is used in various real-world applications, from NASA telerobots that deliver payloads[1], to surface finish applications such as grinding and polishing[2]. Robotic arms are even being added to aerial robots to allow for additional degrees of freedom[3].

II. METHODS

A. Forward Position Kinematics

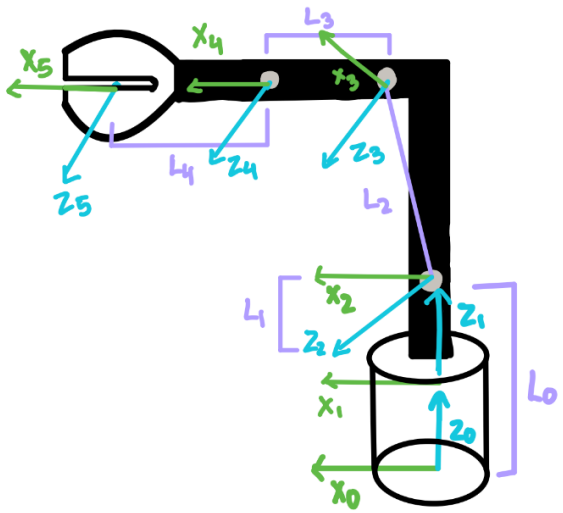

Fig. 1. References Frames 0-4

[1]Nguyen, Charles C., and Farhad J. Pooran. "Dynamic analysis of a 6 DOF CKCM robot end-effector for dual-arm telerobot systems." Robotics and Autonomous Systems 5.4 (1989): 377-394.

[2]Yang, Guilin, et al. "Kinematic design of a 2R1T robotic end-effector with flexure joints." IEEE Access 8 (2020): 57204-57213.

[3]Bodie, Karen, Marco Tognon, and Roland Siegwart. "Dynamic end effector tracking with an omnidirectional parallel aerial manipulator." IEEE Robotics and Automation Letters 6.4 (2021): 8165-8172.

L0	36.076
L1	60.25
L2	130.231
L3	124
L4	133.4

TABLE I
LINK LENGTHS(MM)

Link	θ	d	a	α
1	0°	36.076 mm	0	0°
2	θ_1°	60.25 mm	0	-90°
3	$-79 + \theta_2^\circ$	0	130.231 mm	0°
4	$79 + \theta_3^\circ$	0	124 mm	0°
5	θ_4°	0	133.4 mm	0°

TABLE II
DH PARAMETERS

After finding the DH parameters for each link, the following homogeneous transformation equation was used to find T_i^{i+1}. This calculation is done in the function `dh2mat()`.

$$T_i^{i+1} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & cos(\theta_i)\cos(\alpha_i) & cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To calculate the homogeneous transformation matrix from the robot base frame to the end-effector, the transformation matrices between each link were multiplied. This calculation is implemented in `fk3001()` to calculate the orientation and position of the end-effector with respect to the base frame.

B. Inverse Position Kinematics

To calculate the inverse position kinematics, joint variables θ_1, θ_2, θ_3, θ_4 were solved for using the position and orientation of the end-effector. The inverse kinematics was solved geometrically in `ik3001()`. To account for the workspace limitations of the robot, an error was thrown if the end-effector pose was greater than the length of the arm fully extended or if it was below the z-axis. This safeguard ensures that the robot operates within its workspace and prevents any unintended configurations.

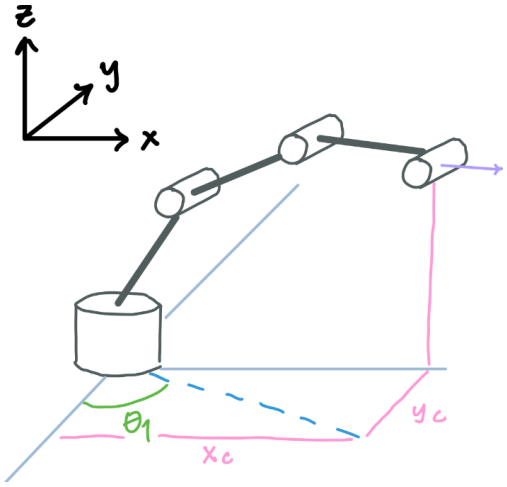

Fig. 2. Solving for θ_1

$$r = \sqrt{x_c^2 + y_c^2}$$

$$\cos(\theta_1) = \frac{x_c}{r} = \quad D$$

$$\sin(\theta_1) = \pm\sqrt{1 - \left(\frac{x_c}{r}\right)^2} = \quad \pm C$$

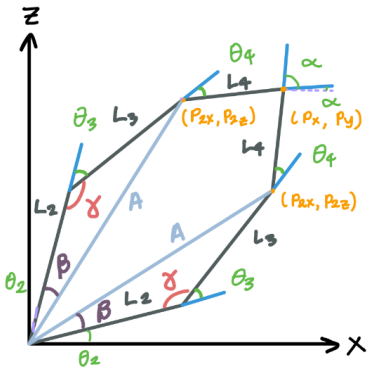

Fig. 3. Solving for θ_2, θ_3, θ_4

$$A = \sqrt{p_{2x}^2 + p_{2z}^2} =$$

$$A = \sqrt{(P_x - L_4\cos(\alpha))^2 + (P_z + L_4\sin(\alpha))^2}$$

$$b = \frac{L_2^2 + L_3^2 - A}{2 \cdot L_2 \cdot L_3}$$

$$c = \frac{L_3 \cdot \sin(\gamma)}{A}$$

$$\gamma = \text{atan2}\left(\sqrt{1-b}, b\right)$$

$$\beta = \text{atan2}\left(\sqrt{1-c}, c\right)$$

$$\sigma = \text{atan2}\left(P_{2z}, P_{2x}\right)$$

$$\theta_1 = \frac{x_c}{r} = \quad D$$

$$\theta_2 = 90 - \beta - \gamma$$

$$\theta_4 = \alpha - \theta_2 - \theta_3$$

C. Forward Velocity Kinematics

The manipulator Jacobian was calculated to compute the linear and angular velocities of the end effector given the joint velocities. The linear and angular velocity components of the Jacobian matrix were solved for separately. The Jacobian function in MATLAB was utilized to take the derivative of the x,y, and z positions of the end-effector with respect to each of the four joint variables. The row results of these calculations were added to the upper half of the Jacobian to calculate the linear velocities. To complete the bottom three rows of the Jacobian, the transformations from base 0 to 1, 0 to 2, 0 to 3, and 0 to 4, were calculated. The third column of each transformation was inserted into the bottom half of the Jacobian matrix. This is because the third column of the individual transformations represents the rotation about the z-axis, which ultimately contributes to the angular velocity of the end-effector. The Jacobian matrix was calculated symbolically and then copied and pasted it into the `jacob3001()`. An emergency stop was set to prevent it from going into a singular configuration (where the determinant of the Jacobian is 0). The function `jacob3001()` considered the determinant when it changed configurations, and when it reached a range near 0, it threw an error and stopped plotting. Having such an emergency stop is important because the robot loses one or more of its degrees of freedom and its movements can become fast enough to cause damage to itself. The function `pwdot()` takes in the current joint angles and instantaneous joint velocities and uses the Jacobian to find the end-effector's instantaneous velocities.

D. Camera Calibration

The image processing toolbox in MATLAB was utilized to calibrate the camera to detect the checkerboard and obtain the camera's intrinsic parameters. First, a set of 40 images was taken and the x-axes, y-axes, and origins of the checkerboards were identified for each. Then, fisheye calibration was run on a set of 20 sub-images until a mean reprojection error below 1.00 was obtained. The camera parameters were exported to a separate script to be readily used later.

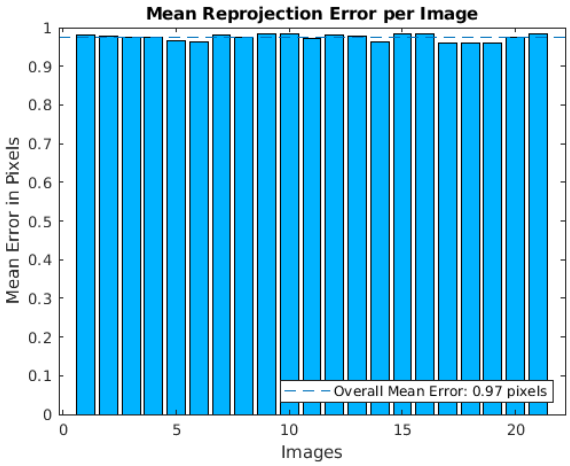

Fig. 4. Reprojection Errors

```
World Points is 40 x 2
Image Points is 40 x 2
The checkerboard is 5 squares long x 11 squares wide
Saved Camera Parameters to camParams.mat
Ready for photo
```

Fig. 5. Camera Calibration Output: Checkerboard Detected

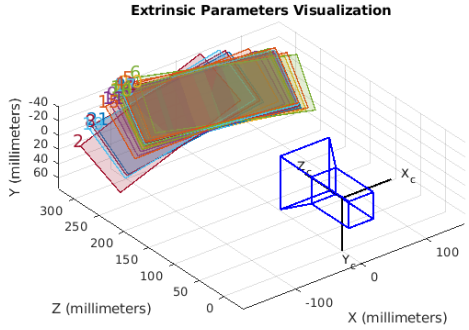

Fig. 6. Extrinsic Parameters Visualization of Camera Object

E. Camera Robot Registration

Running the method `getCameraPose()` determines the extrinsic calibration parameters of the camera and outputs a rotation matrix and a position vector. The method `pointsToWorld()` transforms a point in pixel coordinated from the image frame to a point in the checkerboard frame using the rotation matrix and translation vector.

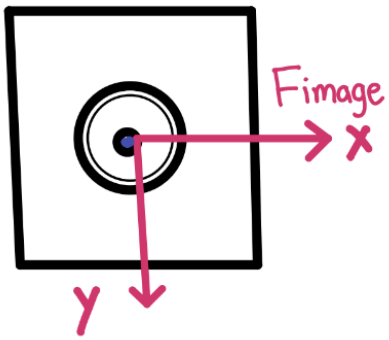

Fig. 7. Image Frame

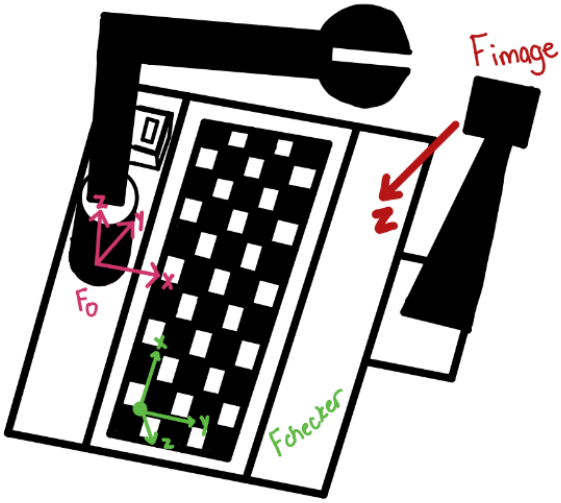

Fig. 8. Checkerboard Frame

Measured distance from $F_{\text{checker\_x}}$ to $F_{\text{robot\_x}}$: 113 mm
Measured distance from $F_{\text{checker\_y}}$ to $F_{\text{robot\_y}}$: 83 mm

θ	d	a	α
90°	113 mm	-83 mm	180°

TABLE III
ROBOT FRAME TO CHECKERBOARD FRAME TRANSFORMATION

$$R_{0_{\text{checker}}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$t_{0_{\text{checker}}} = \begin{bmatrix} 113 \\ -83 \\ 0 \end{bmatrix}$$

$$f_{02_{\text{checker}}} = \begin{bmatrix} R_{0_{\text{checker}}} & t_{0_{\text{checker}}} \\ 0 & 1 \end{bmatrix}$$

$$\text{checkboardpt} = \begin{bmatrix} \texttt{pointsToWorld}(x) \\ \texttt{pointsToWorld}(y) \\ 10 \\ 90 \end{bmatrix}$$

$$\text{baseframept} = \text{inv}(f_{02_{checker}}) \cdot \text{checkboardpt}$$

`checkboardpt` stores the point obtained from the image frame that is converted to be in the checkerboard frame. The x and y components are obtained from the function `pointsToWorld()`, and the z component is the radius of the ball, which is 10 mm. To convert `checkboardpt` into `baseframept`, the point represented by the base frame of the robot, `checkboardpt` has to be multiplied by the inverse of the homogeneous transformation matrix from the robot base frame to the checkerboard. `checkboardpt` has an α value of 90 degrees to ensure the end effector orientation is pointed downwards when placing the balls. Now that the robot has a point in reference to its base frame, this point can be passed to `ik3001()` to calculate the necessary joint angles to allow the end effector to reach this position and orientation.

F. Object Detection and Classification

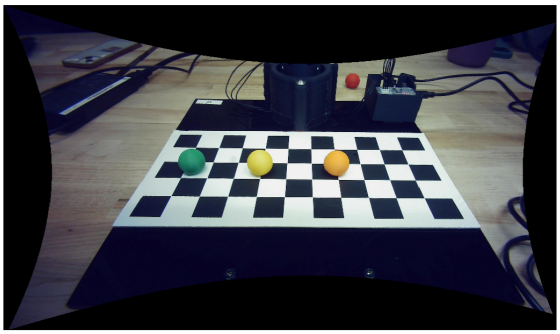

Fig. 9. Undistorted Image

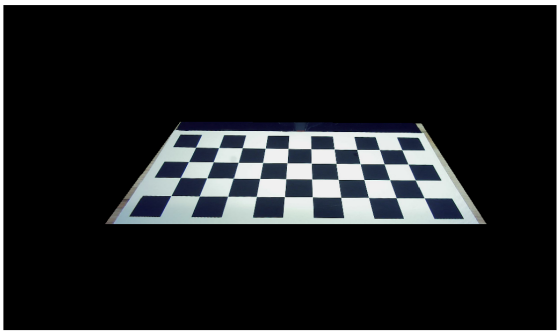

Fig. 10. Checkerboard Mask

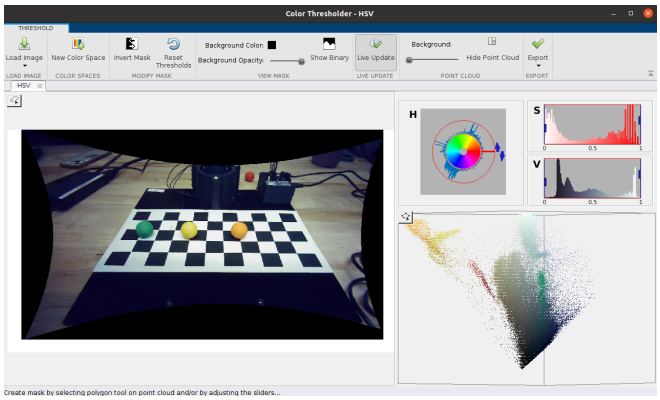

Fig. 11. Camera Threshold Toolbox

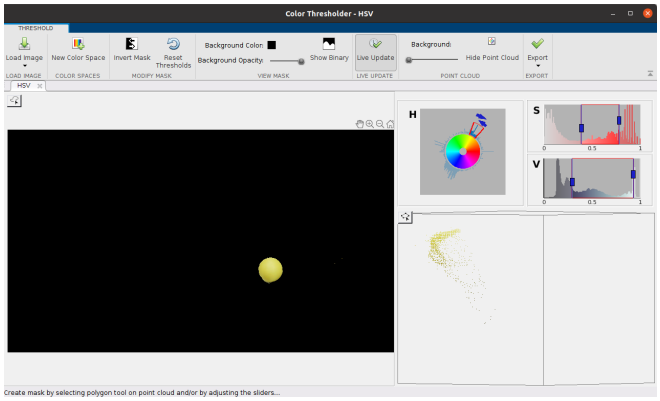

Fig. 12. Use of HSV to Filter Yellow

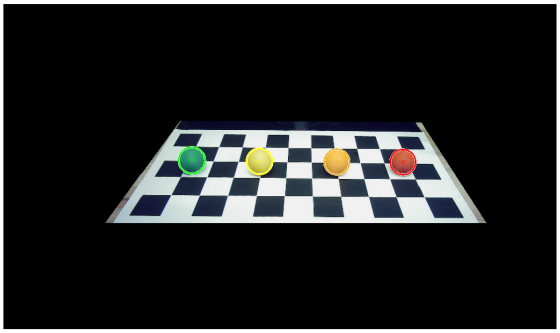

Fig. 13. Ball and Color Identification

A series of masks was first applied to detect the balls within a captured image and identify each of their colors. After removing the image's distortion, a mask was applied to isolate the checkerboard, seen in Figure 10. Next, a series of color masks was applied to find each ball's color. To create a mask for a single color, we utilized MATLAB's built-in camera color toolbox. With the toolbox, we took a captured image and filtered it for the selected color using the lasso select. See Figures 11 and 12. This filter was then used to create the function needed to create the color mask. This process was done for the colors red, yellow, orange, and green. Following this, `imfindcircles()` was used to detect the balls, which allows circular shapes to be identified within images. For `imfindcircles()` to work more consistently, some of its parameters were edited including radius, edge threshold, and sensitivity. The function `viscircles()` was also used which stored the ball center locations within matrices corresponding to each color. It also allowed us to plot the centers of the balls on the captured image to visualize whether or not the camera was locating the balls correctly. See Figure 13 for the final result.

G. Object Localization

To account for the error of the coordinates returned from `pointsToWorld()` due to the camera projection of the height of the ball, it was necessary to calculate the distance between the camera's prediction of the centroid and the actual centroid of the ball. Since we knew the distance of the camera to the origin, the camera's height, and the ball radii, similar triangles were utilized to calculate the distance between the predicted and actual x and y values of the centroid of the ball.

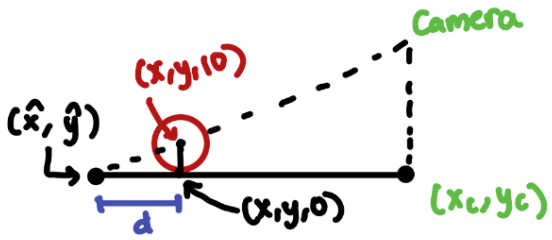

Fig. 14. Correction for Projection on Checkerboard

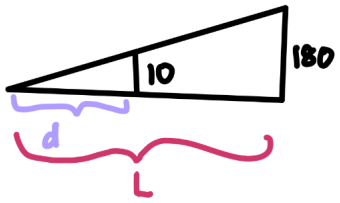

Fig. 15. Solving for Length d Using Similar Triangles

Distance of camera from origin of checkerboard: 365 mm
Height of camera: 180 mm
Radius of ball: 10 mm
$X_c = 365$, $Y_c = 0$
$\hat{x}$, $\hat{y}$ given from `pointsToWorld()`

$$L = \sqrt{(X_c - \hat{x})^2 + (Y_c - \hat{y})^2}$$

$$\left(\frac{d}{L}\right) = \frac{10}{180}$$

$$d = \frac{1}{18} \cdot L$$

$$v = \frac{d}{r}[X_c - \hat{x}, Y_c - \hat{y}] = [x - \hat{x}, y - \hat{y}]$$

$$\frac{d}{r}(x_c - \hat{x}) = x - \hat{x}$$

$$\frac{d}{r}(y_c - \hat{y}) = y - \hat{y}$$

Final correction for camera projection onto checkerboard:

$$x = \frac{1}{18}x_c + \frac{17}{18}\hat{x}$$

$$y = \frac{1}{18}y_c + \frac{17}{18}\hat{y}$$

III. RESULTS

A. System Architecture

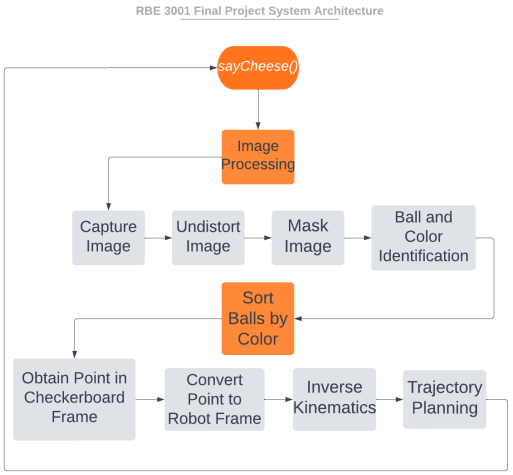

Fig. 16. Final Project Workflow

This project was programmed using MATLAB. Figure 4 provides a flowchart demonstrating the workflow of the sorting system program. The program begins in the script lab5.m, where a single function, `sayCheese()` , is called within a while loop. Once `sayCheese()` is called, image processing begins. First, an image is captured by the camera, which is then undistorted to remove the fisheye lens. This undistorted image is then masked to isolate the checkerboard, allowing for more precise ball and color identification. Once the image processing is complete, a series of functions are called to begin manipulating the robot to sort the balls. First, the centroid locations are obtained in the checkerboard frame and then converted to the robot frame. Once in the robot frame, inverse kinematics and trajectory planning are implemented to move the robotic arm to pick up each ball and sort it according to its corresponding color. `sayCheese()` is called after each time all the balls of a specific color are finished sorting. This ensures that the robot does not try to sort balls that are already set to be removed or are currently being removed. Also, since `sayCheese()` is within a while loop, photos will be continuously taken to check if balls have been added to the board.

To manipulate the robot, several classes were developed. The `Robot` class was primarily used to move the arm and perform forward, inverse, and velocity kinematics calculations. It was also used to convert points from the checkerboard frame to the robot frame. Functions within Robot that were used include:`interpolate_jp()`, `measured_js()`, `jacob3001()`,`convertToRobotPoint()`,`ik3001()`, `run_trajectory()`, and `pickUpDropoff()`. The `Traj_Planning` class was used to create polynomial equations used in `run_trajectory()`. The `Camera` class had several purposes. It was used for capturing images, creating color masks, and recognizing and identifying the locations of the colored balls. It also converted these points to be with respect to the checkerboard frame. Functions within `Camera` that were used include `getImage()`, `sayCheese()`, and `convertToCheckPoint()`.

IV. DISCUSSION

To successfully implement the camera for the robot, we had to ensure that the checkerboard was kept in a well-lit environment. This was because the color masks sometimes did not identify the colors correctly if they were in the shade or if they were too dark. We also implemented the color masks using MATLAB's camera threshold tool. We attempted to modify our color masks so they could see the balls most of the time, but we could not allow for the camera to be overly sensitive. If the camera's sensitivity was too strong, it would identify balls incorrectly or in places where there were no balls. For objection detection, we chose to use the `imfindcircles()` function because it could more reliably locate the locations of the balls than other built-in MATLAB functions, such as region props. Combined with consistent lighting and our tweaked color masks, we eventually were able to detect the colored balls every time we ran the program. We began using the same seat in the lab to keep all of these conditions consistent.

For moving the end effector to pick up the ball, quintic trajectories were used as opposed to cubic trajectories. We found in Lab 4 that quintic trajectories allow for smoother travel, greater accuracy while moving, and reduced vibrations in our arm. To calculate the offset of the ball, we originally applied our calculations to the point directly obtained from `pointsToWorld()` and then used the corrected point to determine the position of the ball in the robot frame. While testing, we noticed the robot was not picking up the ball accurately, so we changed our calculations to account for

the offset after it was already converted to a point in the robot frame. After implementing this change, our robot was able to accurately account for the projection of the ball onto the checkerboard and pick it up. We noticed while picking it up, due to the amount the gripper closed, the ball would sometimes be squeezed with excessive force and eject out from the gripper. To resolve this issue, we changed the angle the gripper opened to to be -35 degrees and changed what it closed to to be 50 degrees.

We also completed both extra credit portions, which included picking up non-circular objects and dynamically tracking objects in the robot's task space. The masks we had created using the color threshold app in MATLAB allowed for the detection of objects in a similar color range, therefore the camera was able to identify a red marker cap and rubber duck for the robot to pick and place. To dynamically track objects, we moved `pickUpDropoff()` to the `Camera` class and continuously called `sayCheese()` in a while loop while the robot was running in its trajectory. This allowed the robot to take images at a rapid rate and track the ball while it was being moved. To prevent the robot from crashing into the ball, we adjusted the z-axis so that it instantaneously followed the ball at a safe distance from above.

V. CONCLUSIONS

The final project allowed us to successfully implement a pick-and-place sorting system on the OpenManipulator-X robotic arm by combining camera vision with concepts from previous labs and lectures. In previous labs, we performed forward and inverse kinematics with trajectory planning to manipulate the robot. We used these functions and calculations in conjunction with information from the camera, which was used to capture images and identify the colors and locations of balls by utilizing a series of masks. Obtaining the locations of the balls from the camera was critical in manipulating the arm to the correct positions. The accumulation of MATLAB functions and course materials helped us prepare for this final project.

APPENDIX

GitHub Repository
Team 5 Video

CONTRIBUTIONS

TABLE IV
TEAM CONTRIBUTIONS

Introduction	Suki
Methods	Suki, Istan
Results	Suki, Istan
Discussion	Suki, Istan
Conclusion	Istan
Video	Tracy
Experimentation	All
Planning	All
Analyzing Results	All
Coding	All

REFERENCES

[1] Nguyen, Charles C., and Farhad J. Pooran. "Dynamic analysis of a 6 DOF CKCM robot end-effector for dual-arm telerobot systems." Robotics and Autonomous Systems 5.4 (1989): 377-394.
[2] Yang, Guilin, et al. "Kinematic design of a 2R1T robotic end-effector with flexure joints." IEEE Access 8 (2020): 57204-57213.
[3] Bodie, Karen, Marco Tognon, and Roland Siegwart. "Dynamic end effector tracking with an omnidirectional parallel aerial manipulator." IEEE Robotics and Automation Letters 6.4 (2021): 8165-8172.

[1]This final project was completed with the gracious support of all the lab staff and Professor Agheli.