Robot Pick and Place System*

Elliot G. Reese, Diego G. Pena-Stein, and Kang Zhang

***Abstract* — This report details our team's approach to implementing an automated ball-sorting algorithm with the OpenManipulator-X arm and an integrated camera, as well as the cumulative functionality from all our previous efforts with this robot.**

I. Introduction

This project aims to demonstrate our understanding of robot manipulation by moving objects with a robotic arm. We must locate colored spheres and place them in a designated area per color. While this builds off work done in previous labs, we will include documentation as necessary. Baseline principles from previous labs include full implementation of forward, inverse, and velocity kinematics and trajectory planning.

The OpenManipulator-X (OMX) arm is a four-degree-of-freedom (DOF) robotic arm with four rotational joints and an end effector capable of gripping objects. It is controlled by DYNAMIXEL motors capable of position, velocity, and current control.

Our robot has an attached camera and baseplate with a printed checkerboard of known dimensions and locations to locate objects in the real world. This allows for object localization in the robotic arm's workspace by analyzing the object's relative position to the checkerboard and comparing that position to the known frames of the camera and robot base.

II. Methods

A. Computer Vision

Before the robot can start picking up balls, it must first recognize the ball's color and location with respect to the robot base. Using an external fish-eye camera and a checkerboard, the location of each ball can be determined. First, defining the intrinsic camera properties, such as mapping coefficients, distortion center, stretch matrix, and image size, is critical. The properties above can be applied to the original image to create an undistorted checkerboard image.

To calibrate the camera, we used the “Camera Calibration” app. From MATLAB’s Image Processing Toolbox, the “Camera Calibration” app was used to take 40 pictures of the checkerboard in different orientations. Since each of the sizes of each square of the checkerboard is known (25 mm), the camera can approximate its properties. With a distorted image, each point of the checkerboard will be non-linear. However, given the non-linearity, the mapping coefficient can be calculated to distort the image. An example of an image used to calibrate the camera parameters is shown in Figure 1.

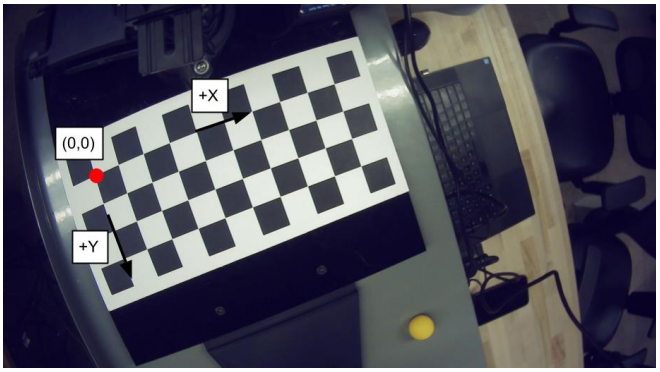

Figure 1: Intrinsic camera calibration to find camera and checkerboard properties.

Before processing calibration images, any partial checkerboards and blurry images were removed to reduce the mean reprojection error. This is critical to represent the camera parameters, as any poor representation of the checkerboard will deviate from the actual camera parameters. Once the camera calibration is completed, an undistorted image can be produced, shown in Figure 2.

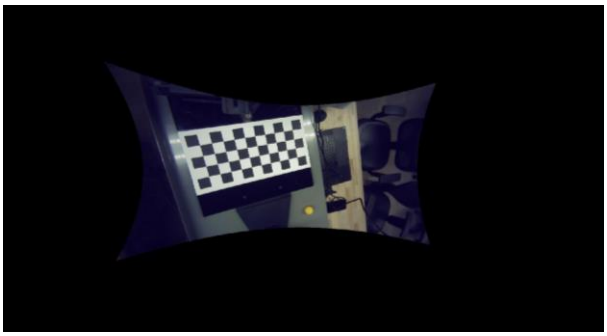

Figure 2: Undistorted image of raw image from Figure 1.

With the camera intrinsics known, a mapping between the camera frame and checkerboard frame (Figure 1) can be made. The camera will be placed on a post (Figure 4) and is in line with the robot’s x-axis (Figure 6). In this location, the camera will take a picture of the checkerboard, calculating its current extrinsic. Understanding the camera's extrinsic properties allows the camera to map from pixel coordinates to real-world coordinates from the checkerboard frame.

*Worcester Polytechnic Institute
Robotics Engineering Program
Course Instructor: Mahdi Agheli
Lab Section: RBE 3001 A’24

Since the checkerboard frame is known with respect to the robot base frame, there is a transformation between the pixel and robot base Equation (1).

$$T_{checkerboard}^{Robot\ Base} = \begin{bmatrix} 0 & 1 & 0 & 80 \\ 1 & 0 & 0 & -112 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

Now, being able to map between pixel and robot base, identifying the pixel centroid for each ball is needed to find its location with respect to the robot base frame. The image taken during the extrinsic calibration will crop future images. A MATLAB function will be used to return the pixel coordinates of the checkerboard. This will allow the image to be cropped with a buffer (make the cropped rectangle bigger) to remove visual noise outside the checkerboard area. Removing as much visual noise as possible is important to reduce the chances of incorrectly identifying balls outside the checkboard area.

Once the camera is fully calibrated, balls can be placed on the checkerboard. A color thresholding mask will be used to identify the distinct colors of each ball, such as red, green, yellow, and orange. A MATLAB app called "Color Thresholding" is used to tune each color mentioned above through hue, saturation, and value (HSV) as a color identifier (Figure 3). The masked image should black out all the colors not within the selected HSV parameter range. This will isolate each ball with the same color from the image, allowing it to be converted to grayscale. Furthermore, the grayscale image can be converted to a binary image (only two colors, black and white).

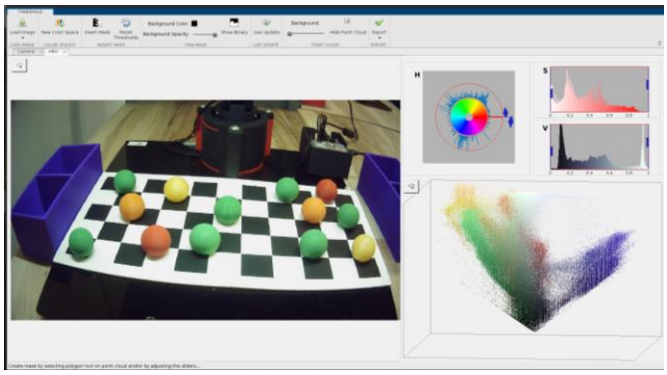

Figure 3: MATLAB app called "Color Thresholding"

At this stage, erosion was tested to filter out further noise left by the color masking. Erosion of the image will make any white areas smaller by "eroding" the shape of each blob. This method would remove any small blobs or large surface area blobs. Erosion was compared without any additional filtering (no erosion or dilation). Lastly, the processed image was analyzed through a block detector. Masking to blob detection would be repeated for each unique color the balls have.

The blob detection algorithm groups pixels into connected clusters and describes each cluster. Any cluster under a certain size is discarded to ignore any slight noise that may have made it through the pipeline. With color masking and blob detector, each ball's properties, such as pixel location, color, and world location, can be identified. The entire image-processing pipeline is shown in Figure 5.

However, the camera registers the ball as part of a flat image and identifies the ball's center point from an offset angle. When the robot calculates the task space coordinates of the ball, it uses the projection of that center point of the ball as projected along the axis to the camera lens as depicted in Figure. To account for this and ensure that the robot moves to the true center of the ball, we calculated the angle between the baseplate and the axis from the camera lens and the center of the ball. Utilizing this angle and the known height of the ball, we determined the offset of the actual location as compared to the position given by the intrinsic function along the axis of the camera. We then converted the offset to the world frame by calculating the angle from the world x-axis of the ball center-camera lens axis and breaking the offset down into the constituent vectors in the world x-y plane. The transformation in Equation (1) can be used with the offsets applied.

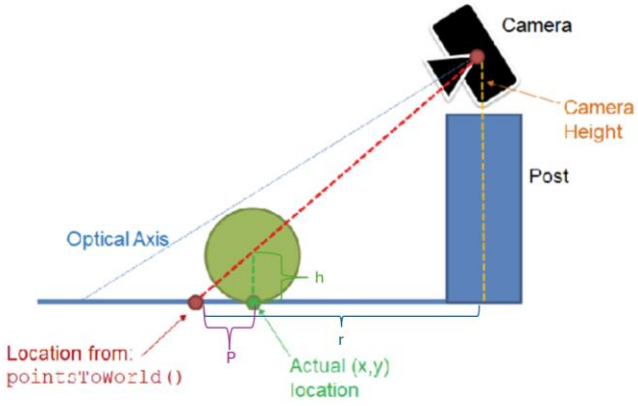

Figure 4: Ball localization diagram.

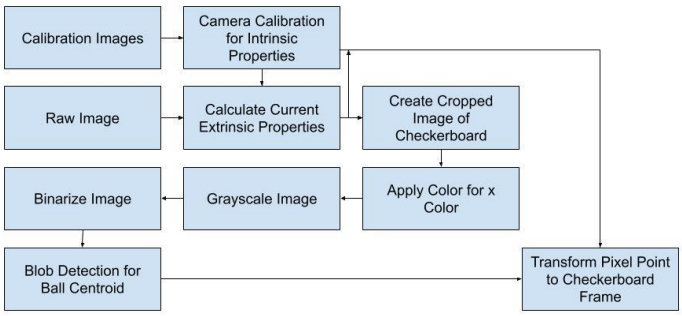

Figure 5: Diagram of the image processing pipeline.

B. Forward Kinematics

The first step to solve this problem is to calculate where the robot is in the physical world. Precisely, we must move from joint space (q) to task space (p). Joint space is a 4x1 vector comprised of each joint's offset from the home

position. Each value is an angular offset (degrees or radians) because the OMX has no prismatic joints. Task space is the traditional cartesian 3x1 vector of x, y, and z world coordinates or an offset from the base frame (also known as the origin).

To translate, we need a transform matrix to define the relationship between joint and task space.

$$p = T_{base}^{ee}(q)$$

p is the 3x1 task space vector

$$p = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

q is the 4x1 joint space vector

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}$$

and T is a 4x4 transformation vector that defines the relationship between the two. The last column is extracted to find the position vector.

The Denavit-Hartenberg (DH) convention provides a standard way to assign reference frames to calculate the transformation matrices for joints on a robot quickly. This, in turn, is to know the position of the end effector quickly. Following the DH convention, we devised the following reference frames for each robot joint. The transforms to each sequential frame and the link lengths are used to generate the DH table. A transformation matrix can be generated between two joints using each value and variable in the table. Multiplying these transformation matrices can build the transform matrix from the base frame to the end effector.

$$T_{base}^{ee} = T_0^1 T_1^2 T_2^3 T_3^4 T_4^5$$

However, to graph each frame, joint, and linkage, it is necessary to keep track of each joint's transformation matrix. The position vector of the transform matrix can be extracted for the position of the frames and joints, and the rotation matrix can be extracted to show each axis of the reference frame at that joint.

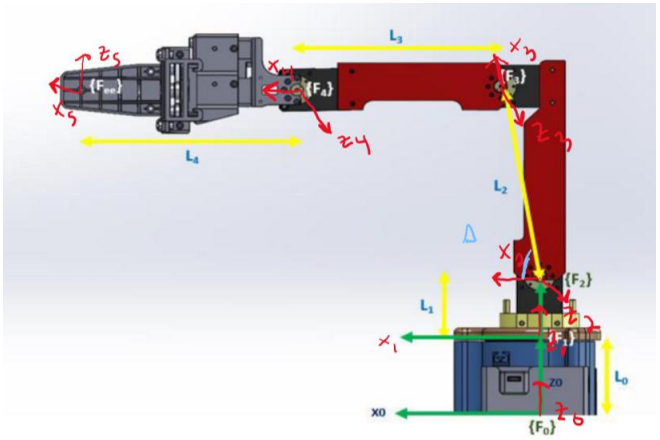

Figure 6: Frame Representation of OMX.

Table 1: Denavit–Hartenberg Table for the OMX.

Frame	*θ°*	*d (mm)*	*a (mm)*	*α°*
F1-2	θ_1	96.326	0	-90
F2-3	θ_2-79.380	0	130.231	0
F3-4	θ_3+79.380	0	124	0
F4-5	θ_4	0	133.4	90

For ease of use, we defined the first (0th) frame to be the same as the one at the first joint, which gives a transformation matrix equal to the 4x4 identity matrix.

Transform	Matrix
T_0^1	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
T_1^2	$\begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & -1 & 0 & 96.326 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
T_2^3	$\begin{bmatrix} \cos(\theta_2 - 1.385) & -\sin(\theta_2 - 1.385) & 0 & 130.231\cos(\theta_2 - 1.385) \\ \sin(\theta_2 - 1.385) & \cos(\theta_2 - 1.385) & 0 & 130.231\sin(\theta_2 - 1.385) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
T_3^4	$\begin{bmatrix} \cos(\theta_3 + 1.385) & -\sin(\theta_3 + 1.385) & 0 & 124\cos(\theta_3 + 1.385) \\ \sin(\theta_3 + 1.385) & \cos(\theta_3 + 1.385) & 0 & 124\sin(\theta_3 + 1.385) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
T_4^5	$\begin{bmatrix} \cos(\theta_4) & 0 & \sin(\theta_4) & 133.4\cos(\theta_4) \\ \sin(\theta_4) & 0 & -\cos(\theta_4) & 133.4\sin(\theta_4) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

C. Inverse Kinematics

In addition to knowing where the end effector is in the task space, we need a way to translate from a desired position in the cartesian task space into angular joint space that the robot can move to. This is done using inverse position kinematics.

Since multiple configurations result in the same end effector position, an additional constraint that must be used is the orientation of the end effector, α.

Only the first joint rotates about the Z-axis, and with the given end effector pose, the angle of this joint can be calculated independently from the other three joints. This means the end effector, and consequently the other three joints, will all be in the same plane (r-z, as seen in Figure 7). Moving joint 1 to align the end effector in this plane will have two collinear, inverse solutions for q_1.

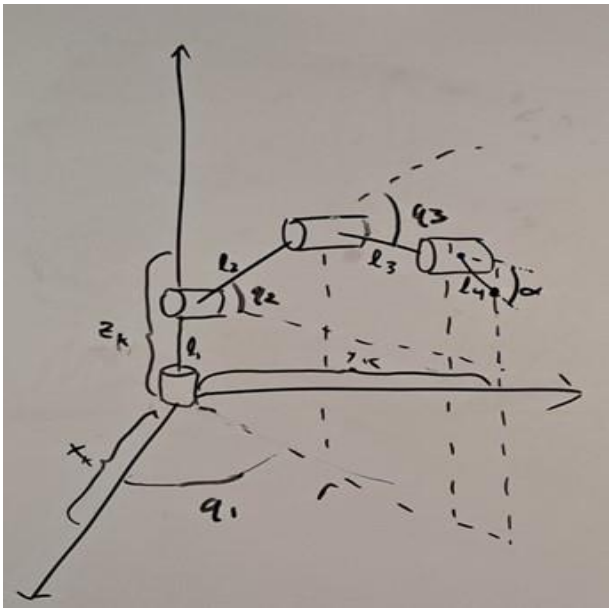

Figure 7: The r-z plane as seen in 3D space

Once in the r-z plane, the robot arm can be simplified to three degrees of freedom (DOF) in two dimensions. To simplify the problem even further, it can be broken down into a 2 DOF arm (q_2 and q_3) with known end positions (J_2 and J_3) and link lengths (l_2 and l_3); and a single DOF arm with a known link length (l_4) and altitude from the horizontal (-α). With a known end effector position, wrist elevation, and link length (l_4), the position of the fourth joint can be derived. Using the position of the 2nd and 4th joint, the joint angles for q_2 and q_3 can be found.

Using the transforms generated by the Denavit-Hartenberg convention in the last lab, the two joint angles q_2 and q_3 can be solved algebraically as a system of equations. Multiplying the transform matrices at each joint (T_2^3 and T_3^4) and taking the position vector gives a system of equations that calculates the position of joint J_3 with respect to q_2 and q_3. The z equation was omitted because it is a constant 0. Additionally, x and y are replaced with r and z, respectively, because of the switch to the r-z plane, but the math is still equivalent.

$$x = A * \cos(\theta_2) - B * \sin(\theta_2) * \sin(\theta_3) + B * \cos(\theta_2) * \cos(\theta_3)$$
$$y = A * \sin(\theta_2) + B * \cos(\theta_2) * \sin(\theta_3) + B * \sin(\theta_2) * \cos(\theta_3)$$

Squaring and summing each equation gives two possible values for q_3 (Simplified to θ_3 because of angle offset and radian-to-degree conversion).

$$\theta_3 = -\text{atan2}\left(\pm\sqrt{1-\left(\frac{x^2+y^2-A^2-B^2}{2AB}\right)^2}, \frac{x^2+y^2-A^2-B^2}{2AB}\right)$$

These can then be used in the original x and y equations to find solutions of q_2. However, we were unable to correctly solve for q_2 in terms of q_3 and x or y by hand, so we used MATLAB's optimization toolbox fsolve to solve the nonlinear equation and calculate the value for q_2. This is done quickly and efficiently using a Matlab function with a success rate of 97% with 1,000 randomly generated joint configurations. Most failures to correctly calculate IK to FK are due to the elbow configuration being the incorrect up/down, as the pose of the end effector is still correct.

Once q_2 and q_3 are found, q_4 can be calculated as a simple combination of these two values and α, where

$$q_4 = \alpha - q_2 - q_3$$

Now, with a desired end effector position, we must still generate a control signal to move the arm to the desired position. This is done by calculating the 6x4 robot Jacobian $J(q)$. The Jacobian details the contribution of each joint's angular velocity $\dot{q}$ (4x1) in joint space on the end effector in task space, $\dot{p}$ (6x1). This is represented as

$$\dot{p} = J(q)\dot{q}$$

The reason $\dot{p}$Is a 6x1 and not a 3x1 because there are two components to the velocity of the end effector: Linear and angular. These are represented in the top and bottom half of the Jacobian, respectively. The top half of the Jacobian is calculated by taking the partial derivative of the respective column's joint with respect to the x, y, and z equations of the end effector's position vector. This position vector is the first three values of the last column of the transform from the OMX base to the end effector.

$$x = Ac_4(c_1c_2c_3 - c_1s_2s_3) - As_4(c_1c_2s_3 + c_1s_2c_3) + Dc_1c_2 + Bc_1c_2c_3 - Bc_1s_2s_3$$
$$y = Ac_4(s_1c_2c_3 - s_1s_2s_3) - As_4(s_1c_2s_3 + s_1s_2c_3) + Ds_1c_2 + Bs_1c_2c_3 - Bs_1s_2s_3$$
$$z = As_4(s_1s_3 - c_2c_3) - Ac_4(c_2s_3 + s_2c_3) - Ds_2 - Bs_2c_3 - Bc_2c_3 + E$$
$$c_n = \cos(\theta_n), s_n = \sin(\theta_n), A = \frac{667}{5}, B = 124, D = 8\sqrt{265}, E = \frac{48163}{500}$$

To calculate the Jacobian's bottom (angular) half, the z (3rd) column from the transform to from base to end effector of the respective joint in the Jacobian is taken. This is represented by $\hat{z}_n(T_0^i)$, where n is the vector's x, y, or z component, and i is the i^{th} transformation matrix. This leaves the following Jacobian (left symbolic for ease of reference):

$$\begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_3} & \frac{\partial z}{\partial \theta_4} \\ \hat{z}_x(T_0^1) & \hat{z}_x(T_0^2) & \hat{z}_x(T_0^3) & \hat{z}_x(T_0^4) \\ \hat{z}_y(T_0^1) & \hat{z}_y(T_0^2) & \hat{z}_y(T_0^3) & \hat{z}_y(T_0^4) \\ \hat{z}_z(T_0^1) & \hat{z}_z(T_0^2) & \hat{z}_z(T_0^3) & \hat{z}_z(T_0^4) \end{bmatrix}$$

D. *Trajectory Planning*

The camera provides the positional vector of the ball with respect to the camera frame. Since the transformation between the camera frame and the robot's base frame is known, this vector can be transformed into the robot task space. The robot is then given the x and y coordinates of that task space vector and an arbitrary z coordinate as the end position of a cubic trajectory through the task space to position the end effector directly above the ball. Then, the robot is given a path to the true ball position vector, which results in the arm moving directly downwards.

Once the gripper secures the ball, the next trajectory moves the arm to a location outside the checkerboard's masked area based on the ball's color to deposit it. We chose to use polynomial trajectories for our path planning because they combine the ability to control velocity constraints and staged task space waypoints.

E. *State Machine of the Robot*

Upon initialization, the robot performs a calibration of the camera. This involves comparing the distortion of the checkerboard of the current view of the camera to 40 pre-determined photos to establish the camera's position. Once the camera calibration is complete, the robot will utilize the image processing pipelines described in section II.A to detect any balls present on the checkerboard. The robot will move to the home position if no ball is present. Alternatively, suppose one or more balls are in place on the checkerboard. In that case, the robot will execute the planned trajectory to move the arm to the first ball, pick it up, deposit it at the location outside the checkerboard determined by the ball's color, and then repeat from the detection stage.

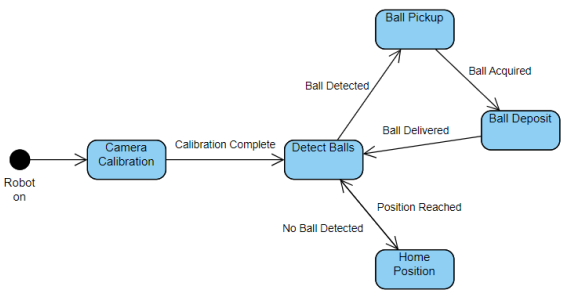

Figure 8: State Machine Diagram

F. *Code architecture*

In addition to the Robot and Camera classes provided, we separated some more complicated methods into their own classes. The cubic and quintic trajectory planning methods and the Jacobian calculations are contained in one class. This is intended to simplify the base Robot class and improve the code's readability. We implemented some of our graphic generators as a class for simplicity, but that class has no bearing on the robot's functionality, only the ease of producing reports.

III. Results

A. *Image Processing*

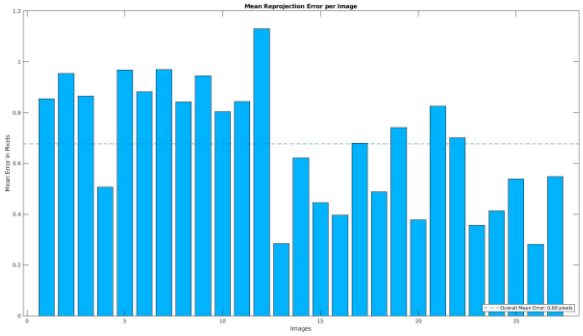

Figure 9: Mean reprojection error per image.

Above, Figure 9 shows the reprojection error when calibrating the camera for its intrinsic properties. The difference between the original raw checkerboard image points and the post-distorted image from an undistorted image calculated the error.

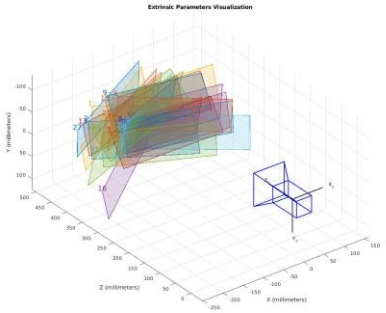

Figure 10: Extrinsic parameters visualization.

Above, Figure 10 This figure shows the extrinsic camera properties for each calibration photo taken. It specifically displays the different orientations and distances the calibration pictures were taken. Figures 11-21 describe the image-processing pipeline from Figure 5.

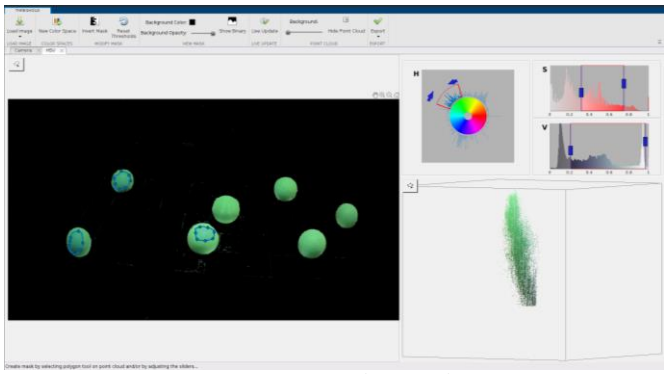

Figure 11: Color thresholding calibration for green for Figure 3.

Figure 12: Raw camera image.

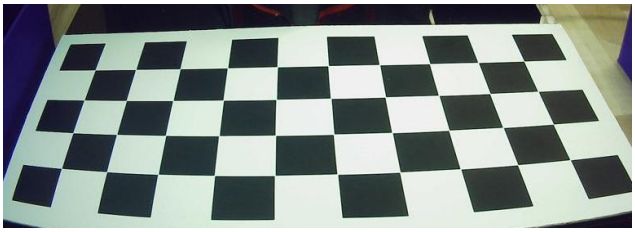

Figure 13: Cropped image for image processing.

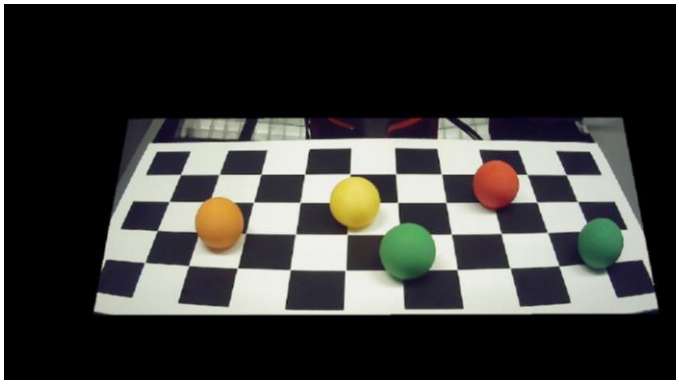

Figure 14: Cropped image to filter noise from outside the checkerboard.

Figure 15: Masked image of cropped checkerboard image (Figure 14). This mask here only displays orange under a specific HSV range.

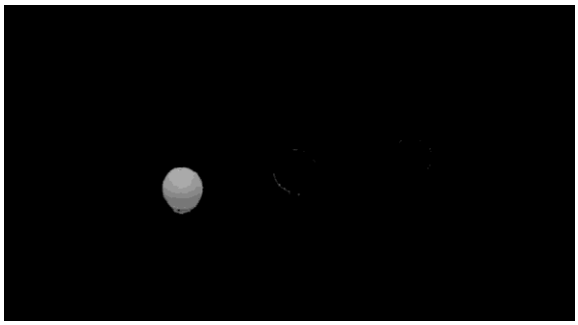

Figure 16: Gray scaled image of Figure 15 to make it black and white.

Figure 17: Binary image of Figure 16.

Figure 18: Finished image processing result of the original checkboard image from Figure 14. It will box the ball and return its color through color masking.

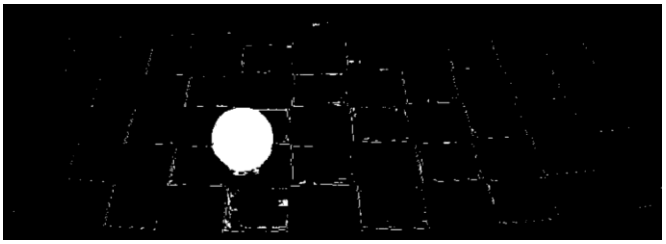

Figure 19: Binary image without erosion.

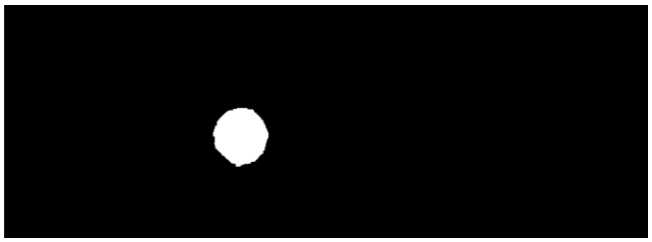

Figure 20: Binary image with erosion.

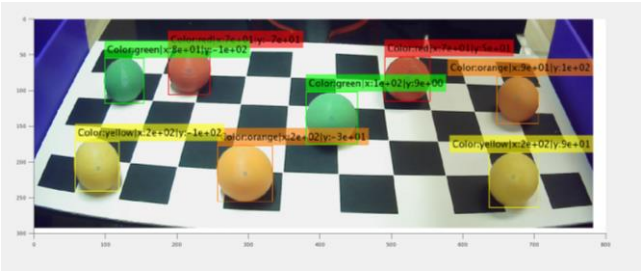

Figure 21: Visual image of color coding and boxing using blob detection.

B. *Forward Kinematic Verification*

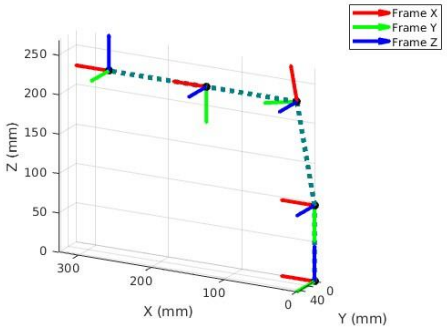

Figure 22: The home configuration of the robot is represented by a stick model.

C. *Inverse Kinematic Verification*

The robot is able to use this color coding and ball identification to pick up each color ball (Figure 23) and deposit them in a specified location (Figure 24).

Figure 23: OMX Picking up a green ball

Figure 24: OMX Depositing a green ball

IV. DISCUSSION

A. *Image Processing*

After completing intrinsic camera calibration, the reprojection error was 0.68 pixels (Figure 9). The reprojection error is the pixel error between the original checkerboard points and the undistorted to distorted checkerboard points. Given the intrinsic camera properties, this value determines how well the undistorted image fits with the original image.

Using MATLAB's color threshold, an approximate value to select specific color features can be made (Figure 11). In Figure 14The image cropping worked well to remove outside visual noise, even if the camera moved during setup. This allows the cropping to be adaptive given each camera calibration to ensure the cropped image is unique for each camera location. However, part of the robot can still be seen, and it has a red 3D-printed part, which is the same color as one of the balls.

However, with proper color masking shown in Figure 15, each of the same-colored balls can be isolated with a black background, making blob detection easier. One confounding factor throughout each experiment was the lighting. If the checkerboard were placed in a dark area, the color masking wouldn't always detect the same-colored balls because the HSV value for each ball would be slightly different due to lighting conditions. The solution to this issue was to use consistent lighting conditions by placing the robot under a light source. This minimized lighting changes throughout the day and kept the HSV value for each colored ball consistent.

Figure 17 and Figure 19 are the final filters (binarized image) applied to each specific color. In the image, there are white pixel artifacts scattered in the image. This is when erosion is applied in Figure 20. However, we noticed that applying erosion in this image was redundant because the blob detection function can filter out any small blobs present in the image. Erosion would only be useful in this case if there were large surface area blobs because it would erode faster than a circle. After conducting blob detection

for each ball, a bounding box is generated, shown in Figure 18.

B. Pixel to ball accuracy

Using inverse kinematics, given the coordinate point of each ball with respect to the robot frame, the robot was able to reach the location of each ball location. In (Figure of the robot on the ball) shows the robot tip aligned with the center of the ball. However, the gripper would sometimes be in different areas relative to the ball because the image processing to detect the center pixel will not be perfectly at the center. The error was caused by a slight inaccuracy in the pixel location of each ball, which was small enough that the gripper could still grab the ball.

V. Conclusion

With the completion of this final project, we will incorporate live image processing to further increase our control of the robot. We implement imaging pipelines to automate the end effector's interactions with specific objects within the task space. The results validate the image processing's effectiveness and accuracy at locating and identifying balls on the checkerboard and planning trajectories for the end effector to pick up any balls and deposit them in their color-specified location.

VI. Appendix

Users	***Implementation***	***Experimentation***	***Documentation***	***Video-Submission***
Kang	33%	33%	33%	33%
Diego	33%	33%	33%	33%
Eliot	33%	33%	33%	33%

VII. References

[1] https://github.com/RBE3001-A24/RBE3001_A24_Team_6/releases/tag/lab5

[2] https://youtu.be/Q2C-VMDZcdc