Lab 5: Culmination of Kinematics and Robot Vision Final Project.

Bryce McKinley, RBE / CS, Mason Sakai, RBE, Carlos Giralt-Ortiz, RBE, Elliot Ghidali, RBE,
Worcester Polytechnic Institute, IEEE

***Abstract*— This is the report for lab 5 by Team 5 from RBE 3001. Its purpose is to test the team's general understanding of the full culmination of topics covered through the 3001 course, including but not limited to, forward kinematics, inverse kinematics, transformation matrices, and a particular emphasis on camera vision / object detection. As a result of its completion, they gained practice implementing and testing all these concepts in a semi-realistic real-world application. Additionally, the team was able to experience the processes of designing and programming an object detection program, thus gaining a more intimate understanding of how they function through the image processing pipeline.**

I. Introduction

The purpose of his investigation was to design, develop, and test a ball sorting robot through the use of the OpenManipulator-X robotic platform, whose foundation was built upon camera vision and object detection. Additionally, this project represents the culmination of an entire term of learning about robot control, including all the knowledge gained from the previous lab assignments, tests, and homework, allowing the team to have an opportunity to practice their developed skills in a complex / semi-realistic real-world scenario. The objectives of this project are as follows:

- Calibrating the provided fisheye lens camera using the provided code and MATLABS built in apps.
- Mapping digital (x,y) coordinates onto the robot's global frame through transformation matrices (specifically one from the camera to the checkerboard, and another from the board to the robot's global frame).
- Detecting the different colored balls through the use of HSV masks, blob analysis, and a few other post processing techniques.
- Adjusting each ball's coordinates based on their height / analyzing how one can both project a 3D object on a 2D plane and vice versa.
- Utilizing forward kinematic and inverse kinematics to pick up and sort the colored balls.

Through the completion of these experiments, the team was able to gain hands-on experience in fully designing, controlling, and testing a robotic manipulator within the context of a semi-practical situation, as well as gain a better understanding of the image processing pipeline / object detection.

II. Methodology

A. Methodology Explanation

For this assignment, the procedure used was provided by Professor Fichera and Professor Aioli in a file called "RBE3001-Lab 5-C24-1". The document itself contained directions explaining what needed to be completed for the lab 5 assignment, information on how to ensure all required materials were prepared, and suggestions on how to approach each open-ended challenge the professors proposed. Below is a synopsis of that report, along with detailed instructions on the methods chosen by the team to complete the project. These directions should be repeatable on one's robotic arm and camera of choice, under the assumption that one has access to the code developed through previous lab assignments, and has properly set up an Ubuntu/MATLAB coding environment.

B. Materials Required

- One OpenManipulator-X robotic arm kit, or another arm of your choice.
- A rectangular checkerboard pattern workspace mat (length 275mm, width 125mm).
- A Micro-USB cable
- A Power cable
- At least one computer with the proper MATLAB and Linux coding environment set up
- Access to the RBE3001 GitHub library on Ubuntu Linux 20.04
- Access to the RBE3001 Lab 5 start code.
- Access to the RBE3001 Lab 4 code from the previous lab assignments.

C. Initial Setup

Before the team could start working on any of the challenges described within the lab, they initially had to prepare the robot and MATLAB workspace. To achieve this, the robot was plugged into the computer using a micro-USB cable and given power through a wall outlet. At this point, depending on if your camera is a separate device from your robotic arm, you may need to connect said camera to the computer and provide it power as well. Then, the previous code from the Lab 4 assignment, along with the provided starter code for Lab 5, was pulled from git into a working computer with the correct Ubuntu/MATLAB coding environment and pushed to its own working branch labeled "Lab 5". Finally, the MATLAB file was modified through the addition of the "Masker.m" class, which was in practice used to locate specific balls based on the Color Thresholder MATLAB app, and will be populated later, and some sort of workspace ("Final.m").

D. Camera Setup & Intrinsic Calibration

Then began the process of preparing the camera for the image processing that came later in the lab. The Camera Calibrator application was downloaded into the digital workspace being used through the Image Processing Toolbox, available on the MATLAB website. Once fully installed, the Camera Calibrator was configured to be able to open and record feeds through the "Camera" tab using the "add images" -> "from camera" settings. An image save location was also determined and chosen through the program's settings, which in the team's case was a specific folder created for this process. Finally, the actual calibration of the camera began with the use of multiple images taken of the

checkerboard workspace and robot from different angles, which were then fed into the camera calibrator app, along with the dimensions of a square of the checkerboard workspace. The app allowed for these photos to be reviewed once submitted, and the team went through them looking for any inconsistencies with the program's detection of the checkboard's x and y axes, removing any they could find.

- An important thing to take into consideration when attempting to repeat this experiment, the program needs about 20 good photographs at this stage to function properly. Thus, if one deletes too many at this stage, the calibration won't work as well as it needs to.

Next, the "Calibrate" function of the app was used, eventually sorting the remaining images into a histogram of reprojection errors, displaying the distance between the checkerboard points detected in each image and the corresponding world points. This graph effectively flags those images with too high of a discrepancy between points, thus allowing the team to once again remove images from the set that would negatively impact their camera calibration, before running the program again to check the remaining result.

- When reviewing this histogram, the team focused on any image with a reprojection error, the actual numerical difference represented in the graph, greater than or equal to one, however depending on the clarity of your camera, you may have to use a different target.

Once the set of photos was deemed acceptable, as seen in Fig. 1, the program "Export Camera Parameters" feature was used to create a calibration script which was generated and saved in the Camera.m class.

E. Camera-Robot Registration

Once the camera was calibrated, the team began working on interpreting pixel frame coordinates from the camera's feed (x,y), into the robotic arm's global frame (x,y,z). They used the provided "getCameraPose()" function, found in the lab 5 starter code, which determines the extrinsic calibration parameters of the camera, based on the script created in subsection D. This first function calculated a transformation matrix between the generated checkerboard's frame and the camera's frame, which was then fed into the "pointsToWorld()" function, provided by the MATLAB image processing toolbox, along with an arbitrary testing point. The second function returned the corresponding (x,y,0) checkerboard frame location.

- The checkerboard frame was generated during the steps in subsection D.

Next, the transformation matrix describing the change from the checkerboard's base frame to the robot's frame was calculated following DH parameters. For the team's particular robotic area, the frame was found to be:

$$\begin{bmatrix} 0 & 1 & 0 & 95 \\ 1 & 0 & 0 & -112.5 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This matrix was multiplied by the resulting (x,y,0) coordinate in the checkerboard frame, to transform it into the robot's global frame.

- To check if this is working properly, follow the optional steps below:
  - Take an image of the task space, display it with the function "imshow", and check 4 additional arbitrary points through the program, and by calculating them manually (using a ruler may be preferred, but knowing the length/width of each square on your checkerboard works too).

F. Object Detection and Classification

To identify the location of objects on the checkerboard, it is first necessary to isolate the field. This serves to prevent objects in the background of the image from interfering with any future image processing. To create a mask to exclude everything but the field, first the corners of the field must be identified, which can be achieved by taking an image from your calibrated camera and hovering over the image at the desired location. These coordinates can then be used in the poly2mask function to create a mask, which can then be applied to individual images with 'image(~mask)'.

To differentiate between the balls on the field, it is necessary to create a mask for each color. This can be done using the Color Thresholder, which should already be installed through the Computer Vision Toolbox. Use the "getImage" function to take a snapshot of the field with a ball of each color on it. Import the image into the Color Thresholder and select a color space. In the case of this project, HSV is the easiest color space to extract the balls with, due to their high saturation in comparison to the checkerboard. Inside the color space, use the saturation slider to remove all low saturations, until only the balls are visible. Use the hue dial to isolate each color in turn, exporting the setting of each as a function. Using these functions, you should now be able to get the black-and-white image of each color completely isolated. Blob analysis is then used to determine the centroid of a ball, as well as generate a box that can be used to surround it. If no blobs are found, return some predetermined failure value for the centroid, such as -1.

- It is recommended to only take the largest blob of each color, to make finding centroids easier later in the process. Once a ball has been removed from the stage, if there are others of the same color, they will then be identified in turn.

A composite image identifying the color of every ball can be created by running each of the color masking functions on a camera snapshot that has been run through the field isolation mask and saving their boxes into a matrix, which can then be passed into the

"insertObjectAnnotation" function, run on the same isolated snapshot, using the rectangle setting.

- Names and colors, in RGB, can be added to the boxes for easy identification as long as they are in the same order of colors that the box matrix is.

To find the pixel coordinates of the centroid of a given color, simply return the value found in the blob analysis.

G. Object Localization

Due to how the camera perceives the object in the physical space the team had to adjust for the ball's height.

- From the camera's elevated and angled perspective point over the balls, as seen in Fig. 6, when it detects a ball's position, thus projecting it onto an (x,y) flat plane, the camera is slightly off, due to said angle and height.
    - The differences between the red line, or the corrected position, and the blue line, or the original recorded position, in Fig. 7 visually display the offset.

This could be done with the following functions, taking checkerPt and *field transformation matrix* as found in subsection E.

$$delta = [checkerPT'; 0] - camera\ position\ matrix$$

$$f = (camera\ Z - ball\ radius)/camera\ Z$$

$$BallPt = camera\ position\ matrix + delta * f$$

$$worldPt = field\ transformation\ matrix * [ballPt; 1]$$

$$robot's\ frame\ position = worldPt(1:3)$$

H. Final Project Challenge

The final part of this assignment required the team to create a state machine that would control the robot to pick up and sort each colored ball. The state machine itself, which can be seen in "Fig. 7", was designed with the following classes

- Waiting: Has two modes based on the perimeter "auto", which determines if all balls are found in pre-set order (o,g,y,r), or if it sorts one type of ball based on user input.
    - If a colored ball is not found, the robot will move to the next color in the hierarchy or detect that there are no balls available and report it, relative to the mode it is in.
- Move: The class that covers all trajectory movement, calling a function "step_traj", which both generates a trajectory and moves the robot along it given a timer. Once each trajectory is completed it switches to another state based on if the robot is picking up or sorting a ball.
    - (Other states calculate the coefficients necessary for the trajectory generation, i.e. the coordinates the gripper needs to go, but "Move" causes the robot to, well, move).
- Move Over: This state moves to a position called "Over Ball" (which is always 5cm above where the ball is found), calculates the forward kinematics before the robot moves to that position, and opens the gripper.
- Move Down: The state that "Move" switches to when picking up a ball, positioning the robot's end effector directly on top of and around the ball.
- Move Up: The state which closes the gripper around the ball and lifts it up to the "Over Ball" position.
- Drop Off: The state which determines the location to drop the ball based on color and moves to it.
- Reset: Finally, this state opens the end effector, removes any variables that were being recorded, moves the robot back to the home position, and switches back to the "Waiting" class.

For the classes that controlled the robot's motion, the forward kinematics, inverse kinematics, and projectile generation implemented were taken directly from labs 2 / 3 respectively. The only real change the team made to the code was using the output of subsection D as the input to the salvaged functions and scripts.

I. Extra Credit: Additional Object

For extra credit the team programmed the arm to pick up an extra object, which in their case was a rectangular eraser. For this extra object, all steps found in subsections F, G, and H were repeated, with careful attention to its HSV values since the eraser's color was very similar to that of the yellow ball.

III. Results

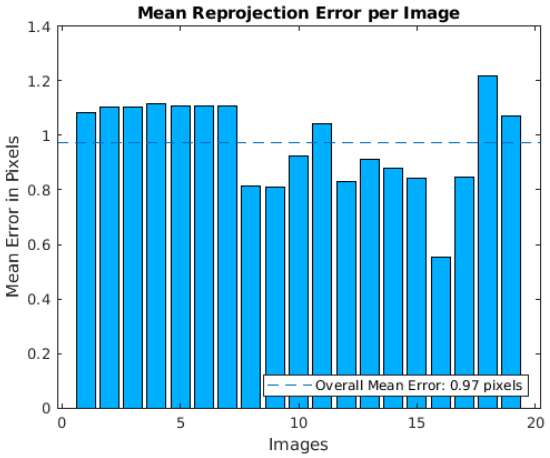

Fig. 1 Histogram Of The Errors From Camera Calibration

Fig. 2 Board Isolation Image Processing Pipeline

Fig. 2.1 The Raw Image From The Camera

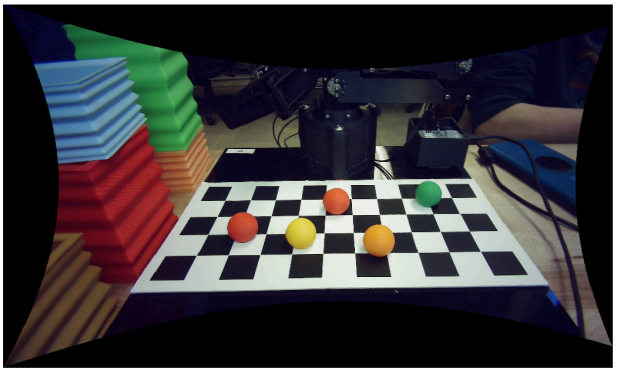

Fig. 2.2 Undistorted Camera Image

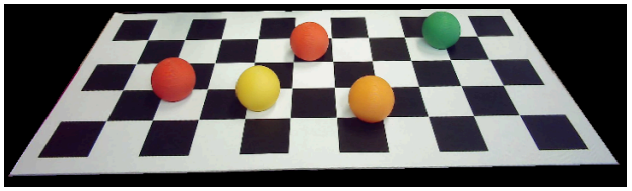

Fig. 2.3 Image With Checkerboard Isolation Mask

Fig. 3 Masker.m Ball Detection Processing Pipeline

The final image processing pipeline for Masker.m. It uses MATLAB's image processing toolbox. This process was done for each color individually; these images show all balls simultaneously with their original colors instead of their separated black and white masks.

Fig. 3.1 Image After Ball Masks

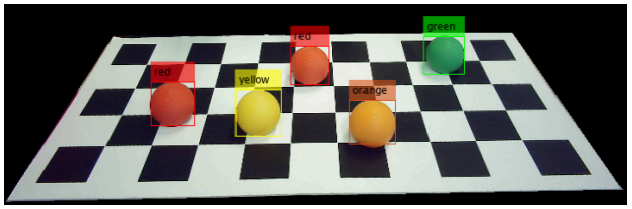

Fig. 3.2 Results Of Blob Detection On The Balls

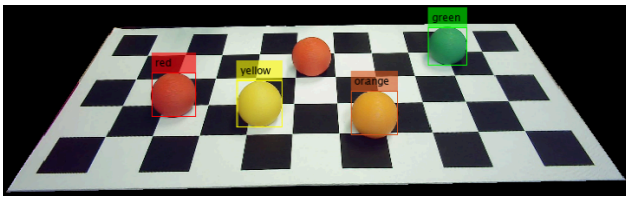

Fig. 3.3 Final Result of the Masker

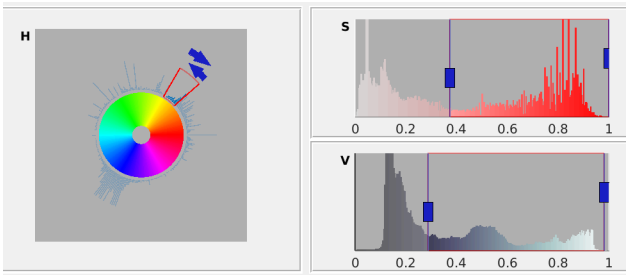

Fig. 4 An Example Filter For The Yellow Ball

Fig. 5 Early Ball Detection Pipeline

A manual, earlier version of the Masker.m ball isolation. It was significantly slower, had less tuned filters, and did not include blob detection. It was however the first ball isolation pipeline we had running.

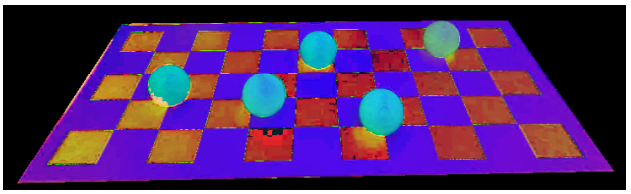

Fig. 5.1 Isolation Image In HSV Color Space

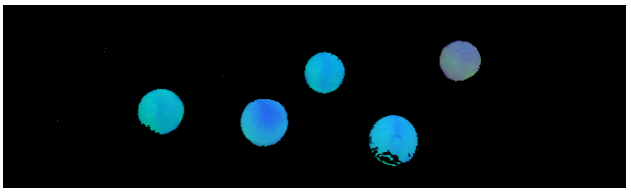

Fig. 5.2 Early Isolation Of The Balls In HSV

Fig. 5.3 Fig. 5.2 Shown In RGB

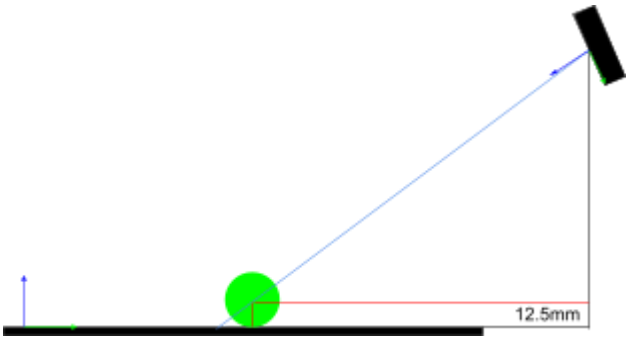

Fig. 6 Geometric Solution Of Pixel To Checkerboard Coordinates

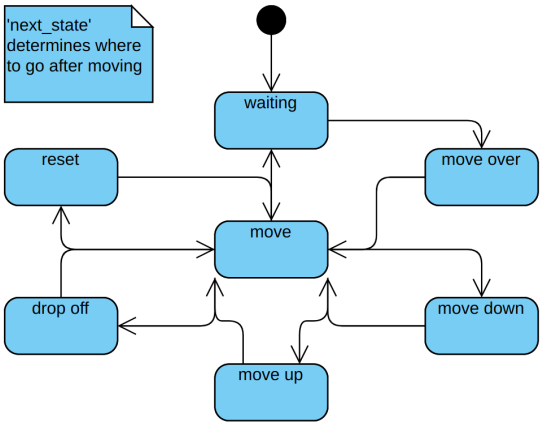

Fig. 7 State Machine Of Robot Operation

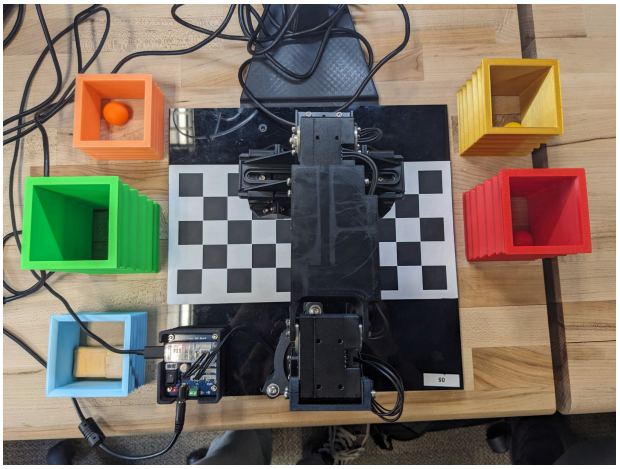

Fig. 8 A View Of The Robot And Sorting Positions

IK

$l_{x1} = \text{sqrt}(p_x^2 + p_y^2) - l_4 * \cos(a)$

$l_{z1} = p_z - l_1 - l_4 * \sin(a)$

$l = \text{sqrt}(l_{x1}^2 + l_{z1}^2)$

$a_0 = \text{atan2}(l_{z1}, l_{x1})$

$\theta_1 = \text{atan2}(p_y, p_x)$

$\theta_2 = \theta_a - a_0 - \cos^{-1}((l_2^2 + l^2 - l_3^2) / (2l_2l))$

$\theta_3 = 180 - \theta_a - \cos^{-1}((l_2^2 + l_3^2 - l^2) / (2l_2l_3))$

$\theta_4 = a - \theta_2 - \theta_3$

IV. Discussion

A. Calibration

The camera was calibrated with 19 images, which were selected for their consistency from 39 total images taken from varying angles surrounding the checkerboard playing field. These images were processed using the Computer Vision Toolbox's Camera Calibration tool, which provides a histogram of the projection error created by each image. After calibration, the mean image projection error was 0.97 pixels (Fig. 1), which is within the margin of acceptable error. With the reference photos and the size of each square on the checkerboard, the Camera Calibration tool was able to accurately determine the location of the camera, as well as undistorting its fisheye lens.

B. Image Processing

Determining the locations of the balls required identifying their coordinates in pixel-space. In order to achieve this, the balls first needed to be isolated. Two different approaches were pursued to achieve this result, both of which started by isolating the checkerboard from the background with a mask (Fig. 2.3).

The first method was done manually, using hand-made thresholds to isolate the balls. The image was converted to HSV, and the range of hue values for a particular color was recorded, followed by the minimum saturation and values for that ball. These recorded values were used to make a mask that took in HSV values and a ball and output a black and white mask for the associated ball. This code was quickly made and shown to work, though the masking function's computation time made it impractical to use it live. It was useful in showing which balls could be problematic in their color, primarily being the confusion of orange and yellow as their hue ranges were connected and their saturation and value ranges were nearly identical.

The second was using the Computer Vision Toolbox's Color Thresholder program, which allows for the modification of images using RGB, HSV, LCbCr, or LAB. HSV was selected for this usage, as it allowed for easy differentiation between the high-saturation balls and low-saturation checkerboard (Fig. 2.3). Each color was then individually separated from the others using its hue (Fig. 3.1). The parameters needed to isolate each ball were saved as functions to the Masker.m class, and then a black-and-white representation of them was processed with blob detection to find only the largest object of each color on the field. Only the largest object was identified to minimize the impact of finding false balls, such as identifying the shadowed part of the yellow ball as orange. This had the effect of ignoring some balls if there were multiple of the same color on the field as seen in Fig. 3.3. Blob analysis returned both the centroids of the balls as well as a bounding box. Once a single ball of each color had been looked for, the bounding boxes were overlaid onto the original image and labeled with their respective colors and names. The updated image was displayed and the pixel coordinates of a selected color were returned.

Although this method located balls very efficiently, it struggled when the lighting on the field was altered. As color identification was dependent on hue, in dimmer light some balls were falsely identified as other colors, and readjusting the masking parameters was time-consuming. To minimize this issue, a strong, consistent light source was always positioned at the same angle above the field, though other ambient light could

still occasionally cause misidentifications, particularly between the yellow and orange balls.

C. Transformation from pixel coordinates to robot frame

The process of calculating the positions of the balls from the pixel coordinates is deceptively simple. MATLAB's pointsToWorld function uses information about the camera to calculate the position of a point on the checkerboard from pixel coordinates. This can then be transformed into world space using the transformation matrix of the checkerboard in the world frame.

This calculation however isn't quite right, as shown in Fig. 6. Since the camera doesn't capture depth information, it can only work in 2D. As such, an assumption has to be made when converting into 3D space, and that assumption is that the point is flat against the checkerboard; which are balls and their centroids are not. This produces an error in the position of the ball, which is estimated to be around 7% of the distance from the camera, or up to 15.9mm when at the farthest point on the workspace. Because of this, an extra step is taken before the transformation from checkerboard to world frame.

For this calculation, 3 significant things are known: the position of the camera, the 2D projection of the ball, and the size of the ball. From this, a corrective factor can be calculated from: (cameraheight - ballradius) / cameraheight. This can be multiplied by the vector between the camera and the 2D ball's projection to get the real position of the ball.

D. Sorting

Once the location of a ball had been identified in the robot's frame of reference, sorting could begin. A state machine (Fig. 7) was used to control the flow of the robot's operation, with the hope of creating a non-blocking control system so that the camera feed could be continuously updated with the location of the balls.

The beginning state of the machine was known as 'waiting' which contained two different operations depending on whether the variable 'auto' was set to true or false. If 'auto' was false, then the system would prompt a user for input in the form of a single letter, either r, y, g, o, or e which is associated with red, yellow, green, orange, or eraser (the extra, non-ball object required for an extra credit sign-off). Once an input was received, it was checked for validity and the first object of the selected color was sorted. If 'auto' was true then the system would check for the existence of objects in the order eraser, orange, green, yellow, then red. To check if an object was present, its associated number was passed into the 'find_all' Masker function, which returned either the colors centroid or -1 if no instance was found. Once a valid centroid was located, the robot would sort the object associated with it, and then return to the waiting state to find the next object. This flow meant that the robot would always sort all objects of a given color before moving on to the next. If there were no valid objects on the field, then the robot would display a message and pause until a key press, allowing the field to be reset.

Once an object was selected, its pixel coordinates were transformed into the robot's frame. It proved necessary to not go directly to the object, as it could cause collisions with other objects on the field if the end effector was too low. To combat this a point 5cm directly above the ball was achieved both directly before and directly after grabbing the object.

Objects were sorted into predetermined spots off the sides of the field based on color (Fig. 8). Placing the balls outside the checkerboard caused them to be excluded alongside the rest of the background thanks to the field mask, preventing them from being targeted again.

One of the greatest hurdles of this approach was changing the *plan_trajectory* function to no longer include a while loop, which was achieved by creating the *traj_step* function. This function consisted of a single round of cubic trajectory calculation and movement and was designed to be called continuously within the state machine until movement was complete. Cubic trajectory was chosen because control of the robot's acceleration was deemed unnecessary in this context.

The state machine handled movement using the *traj_step* function in the 'move' state, which would continually update the robot's joint speeds until the target destination was reached. The 'move' state was used many times during a single round of sorting, in between operations like closing and opening the gripper and calculating waypoints or new trajectory coefficients. Because the state machine needed to know where it was in the process of sorting when leaving the 'move' state, it was necessary to implement a 'next_move' variable which kept track of what state followed the movement (Fig. 7).

Once the state machine was fully implemented, it became apparent that continually updating the ball's locations greatly affected operation, leading to jerky movement and decreased speed. It proved necessary to remove it from the loop, with the ability to re-implement it at the cost of performance.

V. Conclusion

Using the OpenManipulator-X robotic arm and visual processing a ball sorting robot was successfully implemented. To accomplish this goal visual processing was first used to determine the positions of the balls on the field in reference to the robotic arm. Then via known kinematics of the arm, the robot was directed to pick up and move balls from squares on the field in a preset order based on their color and dropped them outside the field.

To successfully find the position of the ball relative to the robot it was first necessary to calibrate the camera to the field. To this end, a checkerboard pattern was used for visibility and multiple photos of the field were taken from several angles. The calibration software assigned

real-world coordinates to the field from the pixel coordinates of the squares. The results were screened for mistakes using their mean reprojection error and the resulting images were processed such that it would be able to assign coordinates in the robot's reference frame when given an image of the field.

To find the location of the balls on the checkerboard the images first had to be processed to a machine-readable form. First, the fisheye lens of the camera used was corrected using a transformation matrix based on its known characteristics. Next using the function, Masker.m from MATLAB's image processing toolbox the checkerboard was isolated from the background, and the balls were isolated from the checkerboard. Now being in a form that could be read by an algorithm; the color and pixel coordinates of the balls were then extracted from the resulting image containing only the colored balls. Using the mapping function that was calibrated earlier the pixel coordinates of the balls were referenced with the pixel coordinates of the checkerboard and using the calibration function, transformed into coordinates in the robot's reference frame. Due to the camera not being directly above the field a significant depth error existed which was corrected via a transformation function involving the balls seen size and the height of the camera.

Having obtained the coordinates of the balls in the robot's frame of reference the arm could then begin the sorting algorithm in one of two modes. In auto mode, the robot begins sorting the balls in the order of eraser, orange, green, yellow, and red. Otherwise when not in auto mode the robot would sort the balls in a user input order. From the coordinates of the ball and the derived kinematics of the arm a cubic trajectory function was generated to move the arm from its current position to a position to grab the selected ball. The ball was then grabbed using the end effector of the arm and a trajectory to the outside of the field was generated and the ball was dropped. This was repeated in the order discussed before until there were no balls of the selected colors left on the field. In this way, the robot was successfully able to sort colored balls on a field.

Appendix A

Team GitHub Repository:
https://github.com/RBE3001-C24/RBE3001_C24_Team05

Appendix B

Video:
https://youtu.be/Yt5aCPd4eT4

Appendix C

Section	Author
Abstract	Carlos Giralt-Ortiz
Introduction	Carlos Giralt-Ortiz
Methodology	Carlos Giralt-Ortiz and Bryce McKinley
Results	Mason Sakai
Discussion	Bryce McKinley
Conclusion	Elliot Ghidali

Acknowledgment

Thanks to Professor Fichera and Professor Aloi for a wonderful term of learning and discovery, as well as all of the amazing student assistants for their unending patience and aid during labs.