RBE 3001 Lab 5 Final Report

Ashe Andrews, Sebastian Baldini, Jakub Jandus

***Abstract*— In this lab, the team used the combined work done in previous labs with new knowledge about computer vision for object detection to program the OpenManipulator-X arm to sort colored balls in its workspace.**

I. INTRODUCTION

A. Background

In previous lab exercises, the team learned skills such as forward and inverse kinematic calculations, trajectory generation, and overall usage of the OpenManipulator-X robotic arm. However, the team had yet to use the camera setup that comes with the robot. Adding camera vision to a robotic system in an industrial setting can make a robot more reliable, consistent, and efficient [1].

B. Motivation

The motivation of this lab was to serve as a cumulative project meant to utilize all of the work done in previous lab exercises this term. This lab also served as an introduction to computer vision concepts such as image processing and color detection. The exercise made use of the camera attachment of the OpenManiulator-X robot setup for color detection as part of a larger sorting task. Completion of this lab shows an understanding of all concepts taught in the RBE 3001 course and how to synthesize them with a robotic arm.

II. METHODS

A. Previous Work

Following procedures from Labs 2, 3, and 4, the team calculated the forward position and velocity kinematics of the robot and its inverse position kinematics. Figure 1 shows the Denavit–Hartenberg frames (also called DH frames) of the on the physical robot. Figure 2 shows the resulting DH table as well as the direction of rotation for each joint. Figure 3 shows the inverse kinematic calculations for the robot with the assumption that the fourth joint is a 1-DOF wrist. Finally, Figure 4 shows the MATLAB code used to calculate the generic Jacobian of the arm for any configuration. The lengths in millimeters of the links the arm are as follows:

- L0 = 36.076
- L1 = 60.25
- L2 = $\sqrt{128^2 + 24^2}$
- L3 = 124
- L4 = 133.4

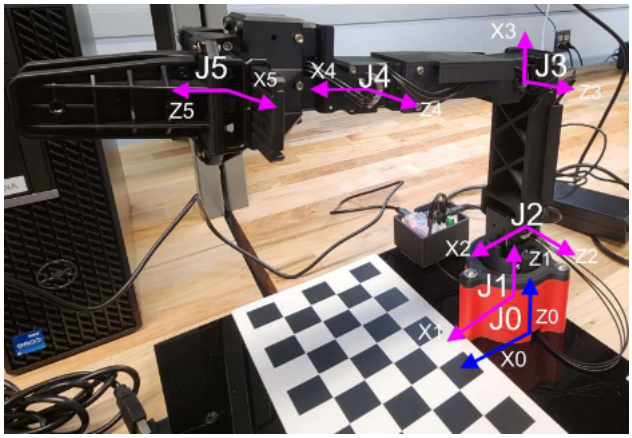

Fig. 1. Frames of each joint of the robot assigned according to the DH convention

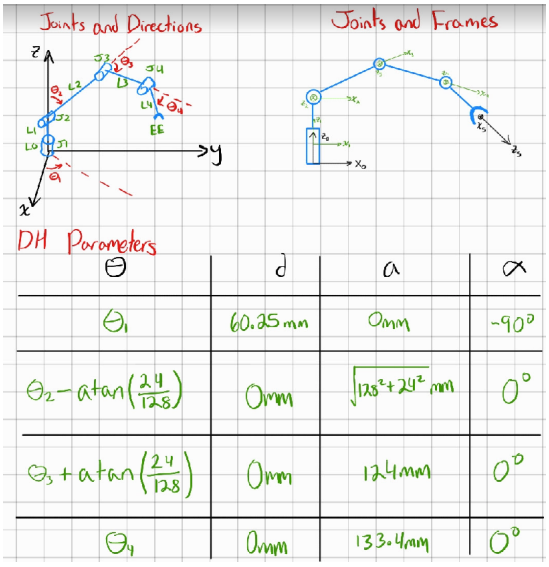

Fig. 2. DH table of the robotic arm and the directions of rotation for each joint

B. Intrinsic and Extrinsic Camera Calibration

To begin working with the camera, the team first performed intrinsic camera calibration with the USB webcam and the checkerboard. The team used MATLAB's Camera Calibrator app from the Image Processing Toolbox and removed the camera from its post to take many pictures at multiple angles for accurate calibration. The team began with roughly 40 images of the checkerboard from various angles and heights, making sure the whole checkerboard was in the frame for each photo. These images were then

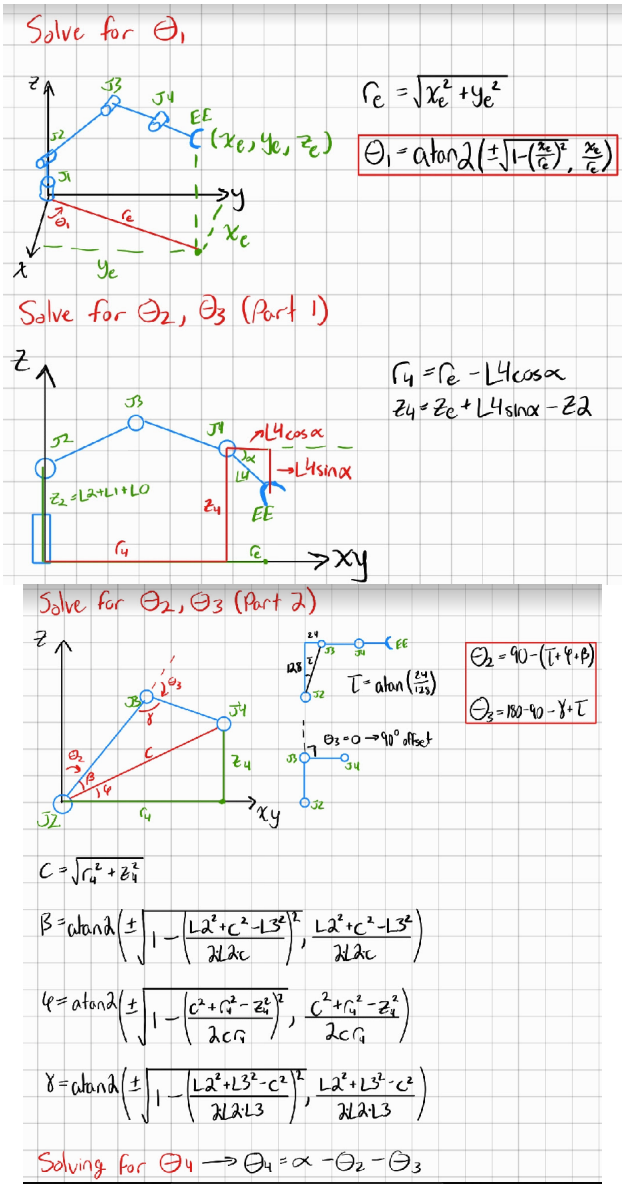

Fig. 3. Calculations of the inverse kinematics of the robotic arm where the position and orientation of the end effector are known

```
% Use dh2fk() to get the symbolic transformation matrix
sym_tm = dh2fk(DH, T01); % symbolic matrix 0 to EE

% Turn that into a numerical transformation matrix
% Recreate the symbols we'll be substituting in
theta = sym('theta%d', [1 size(q, 1)]);
d = sym('d%d', [1 size(q, 1)]);
a = sym('a%d', [1 size(q, 1)]);
alph = sym('alph%d', [1 size(q, 1)]);

% Substitute in for symbolic d, a, and alpha
function finished = subsitute(symbolic)
    finished = subs(symbolic, d(1), DH(1, 2));

    % Iterate through d's
    for i = 2:size(q, 1)
        finished = subs(finished, d(i), DH(i, 2));
    end

    % Iterate through a's
    for i = 1:size(q, 1)
        finished = subs(finished, a(i), DH(i, 3));
    end

    % Iterate through alpha's
    for i = 1:size(q, 1)
        finished = subs(finished, alph(i), DH(i, 4));
    end
end

T = subsitute(sym_tm); % T0EE with only thetas
```

```
% Get x, y, and z values of T0EE
EE_X_sym = T(1, 4);
EE_Y_sym = T(2, 4);
EE_Z_sym = T(3, 4);

X_partial = jacobian(EE_X_sym, [theta(1), theta(2), theta(3), theta(4)]);
Y_partial = jacobian(EE_Y_sym, [theta(1), theta(2), theta(3), theta(4)]);
Z_partial = jacobian(EE_Z_sym, [theta(1), theta(2), theta(3), theta(4)]);

Jo = [X_partial;
      Y_partial;
      Z_partial];

% calculate the J0 of the jacobian
T1 = T01;
T3_temp =  T1(1:3, 3); %Z1 for T01
T2 = T1*dh2mat([theta(1) d(1) a(1) alph(1)]);
T2_temp = T2(1:3, 3);   %Z1 for T02
T3 = T2*dh2mat([theta(2) d(2) a(2) alph(2)]);
T3_temp = T3(1:3, 3);   %Z1 for T03
T4 = T3*dh2mat([theta(3) d(3) a(3) alph(3)]);
T4_temp = T4(1:3, 3);   %Z1 for T04

% use the subsitute function to numerically substitute all variables but thetas
T1_subs = subsitute(T1_temp);
T2_subs = subsitute(T2_temp);
T3_subs = subsitute(T3_temp);
T4_subs = subsitute(T4_temp);

Jo = [T1_subs(1), T2_subs(1), T3_subs(1), T4_subs(1);
      T1_subs(2), T2_subs(2), T3_subs(2), T4_subs(2);
      T1_subs(3), T2_subs(3), T3_subs(3), T4_subs(3)];
```

Fig. 4. MATLAB code used to calculate the Jacobian of the robot

processed through the calibration software to start with an initial calibration profile. The team then used the histogram of image reprojection error to eliminate images with error higher than two pixels; the reprojection error indicates how well the camera calibration found the intersection of the checkerboard squares, and a smaller pixel value indicates the camera sees the intersections very closely to where they actually are. After re-calibrating with the more accurate subset of images, the team went through each image individually to verify that the calibration software recognized the entire checkerboard and assigned the checkerboard frame according to Figure 5. Images that showed a differently assigned frame were removed. After this filtering process, the team performed a final intrinsic calibration and exported the results to a camera calibration script to be called by the Camera class at initialization.

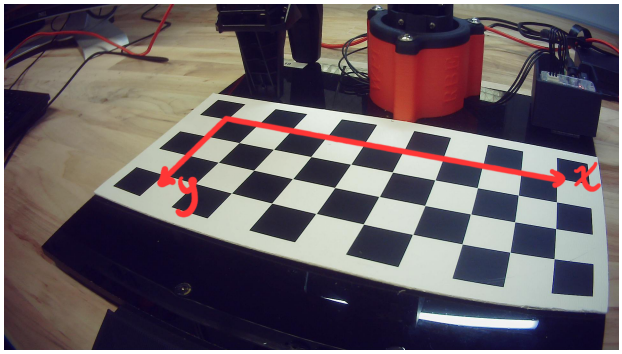

Fig. 5. Designated frame orientation for the checkerboard frame. Images with a different frame orientation were rejected from calibration of the camera

MATLAB calculates the intrinsic calibration of the camera using the length of the squares in the checkerboard and the coordinates of each pixel in the image. Using the checkerboard, the software knows the distinct points at the intersection between squares and the distances between them. It can then use the ratio of the physical distances to the distances in pixel coordinates to calculate the transformation from physical world frame to camera frame.

The team then moved to extrinsic camera calibration using provided template code. To test that the conversion from camera frame to checkerboard frame to robot frame was correct, the team developed a function to get the intersections of each checker in the checkerboard as pixel coordinates. The team then programmed the robot to convert these pixel coordinates to robot task space coordinates using the extrinsic calibration and move to them. After calibration, the robot was able to move the gripper to each intersection with acceptable tolerance.

C. Object Detection and Classification

The team then moved on to detecting the balls and their colors. The team used the HSV color space because of the increased variation between different colors compared to other color space options. Increased variation meant that colors were more distinct from one another, and therefore

they were easier to detect. Additionally, the HSV color space provided some resiliency to changed lighting conditions.

To detect all the balls in the checkerboard, the camera takes an image of the space, which is then undistorted and cropped to only the checkerboard to avoid irrelevant detections. Using HSV color thresholds, the image is masked to contain only balls of the color of interest and then converted to gray-scale for processing. A median filter is applied to remove noise, and then the edges of the balls are eroded and dilated for smoothing and filling the circle shape. The image is then binarized, and the imfindcircles() function is applied to find the centers of the balls. This is done for each color to create an array of points where each row represents balls for a given color.

The team experimented with several filtering and processing sequences to see which combination most consistently ended with a distinct and whole circle in the image. In the final version of the image processing pipeline, the team used the following filters/ processes on the raw camera image to detect the balls and their colors:

1) Remove fish-eye distortion
2) Convert the image from RGB colors to HSV
3) Mask the image to focus only on the checkerboard
4) Mask the image to focus only on the ball of the selected color using HSV thresholds
5) Convert the image to gray-scale
6) Apply a median filter to the image to remove noise
7) Erode edges in the image to remove noise around the ball
8) Dilate edges in the image to fill in the ball
9) Binarize the image so that white pixels represent the ball and black pixels represent everything else

Figure 6 shows the process as a flowchart alongside the resulting images of each filter.

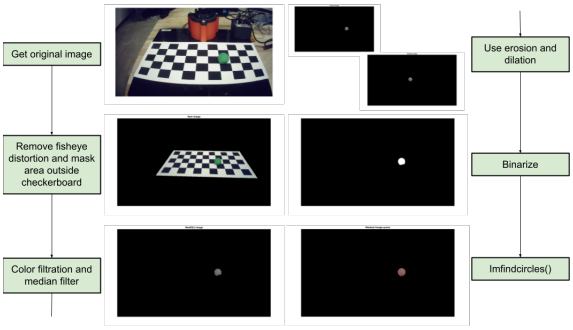

Fig. 6. Flowchart of the image processing pipeline for ball detection with resulting images for each step

The x, y, and z coordinates of the center of the ball is offset form the real world coordinates because the camera sees the closest part of the ball and makes its calculations based on that part of the ball. This is because the camera is 2D and has no depth sensing. To fix this offset, the team included the calculations done in Figure 7 in the code to adjust coordinates involving picking up the balls. Without this correction, the error in the x and y directions would be equal to the ratio of the ball radius and camera height (0.0657) multiplied by the distance in that direction of the ball from the camera, which would be in the range of a few millimeters depending on the ball's location in the checkerboard.

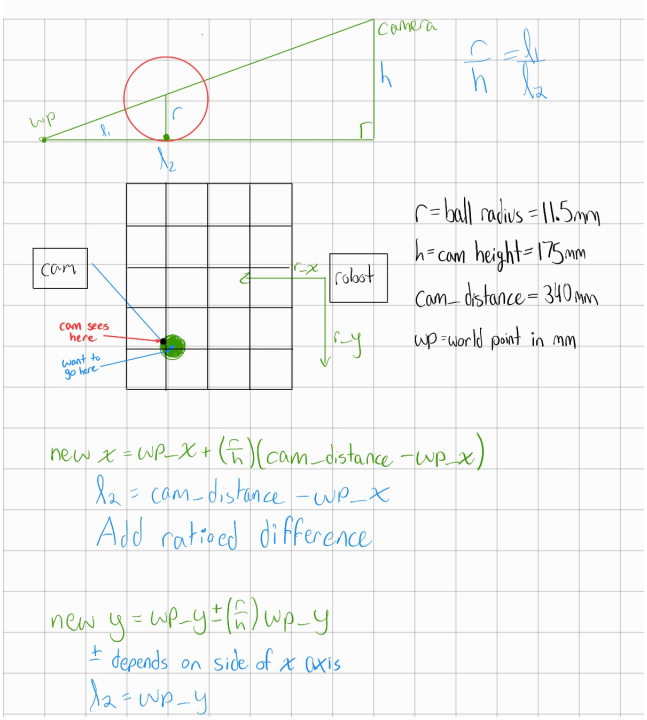

Fig. 7. Calculations used to determine the x and y offset of the center of the ball from the camera image to the robot frame

D. Overall Approach

To sort the balls, the camera detects all balls on field and categorizes them by color. The robot then sorts the balls by color in the following order: blue, yellow, red, green, gray, and orange. After a ball is sorted, the camera takes a new image of the field and re-detects balls in case a ball has been replaced or moved. To exclude sorted balls, the robot places sorted balls in boxes off to the side. Additionally, the camera image is masked to include only what is in the checkerboard: since balls are sorted outside the checkerboard, they are no longer included in the detection sequence once sorted. Figure 8 shows the flowchart of the processes done to sort all of the balls by color.

For moving the robot arm, the team elected to use inverse position kinematics with the interpolate_jp() method. This method was the simplest to implement and requires fewer calculations than using trajectory generation. In Lab 4, the team had also observed that interpolate_jp() and cubic trajectory movements were similarly smooth.

E. System Architecture

Our system's code was split between three main files, Robot.m, Camera.m, and labFinal.m. The Robot class in Robot.m remained largely untouched from previous labs and

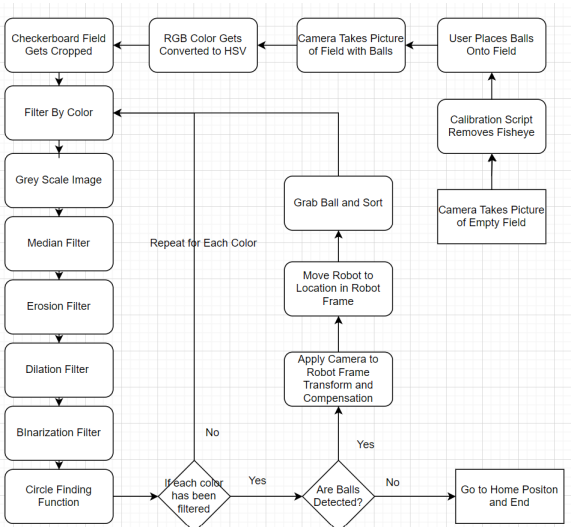

Fig. 8. Flowchart of the entire sorting process completed by the camera and robot

contained the robot control and kinematic functions used to move the robot when sorting the balls. The team added one new function, appro_za() for moving above a ball before picking it up. The Camera class in Camera.m contained provided functions for camera calibration, and the team added several helper functions to the class for image processing, ball detection, and coordinate manipulation. Finally, the labFinal.m file contained the main program to detect the balls and sort them, and it used the methods from the Robot and Camera classes. Figure 9 shows the code structure and describes the helper methods in each file.

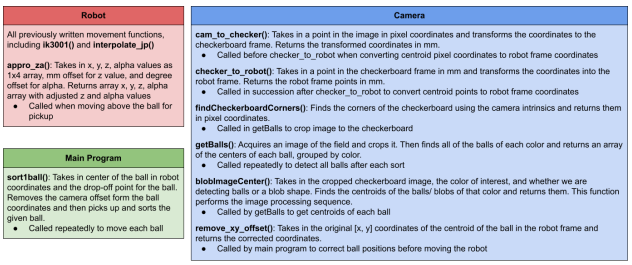

Fig. 9. Class and method structure used for organizing code to complete the lab exercise

III. RESULTS

A. Image Filtering and Object Detection

Figures 10 through 15 show the results of each step of the image processing pipeline.

B. Object Classification in Motion

Figures 16 through 23 show the images after the robot sorts each of seven balls.

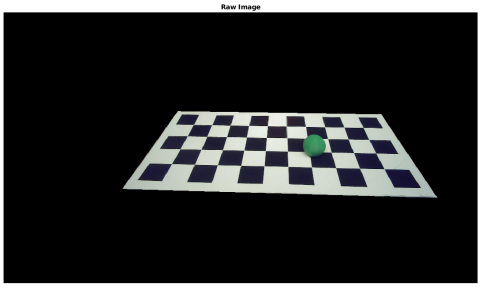

Fig. 10. Initial cropped image of green ball on checkerboard grid

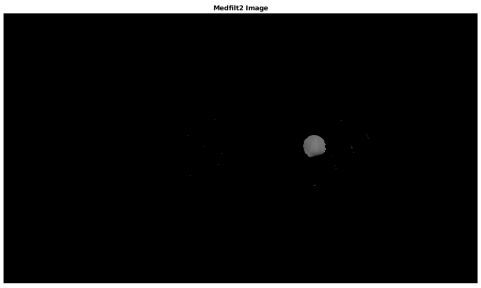

Fig. 11. Color-filtered image after passing through the median filter for noise removal

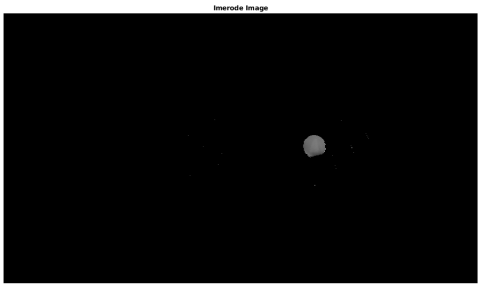

Fig. 12. Color-filtered image after passing through an erosion and a dilation filter

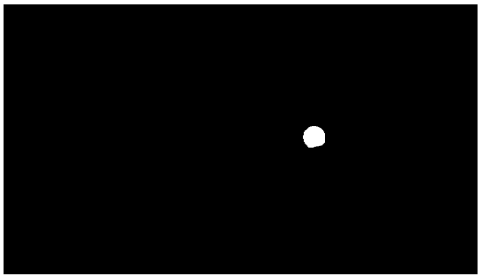

Fig. 13. Binarized image of the ball such that white space represents the ball and black space is everything else

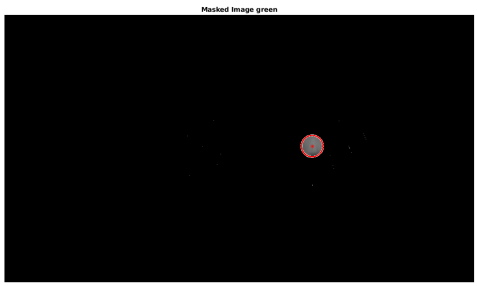

Fig. 14. Results of imfindcircles() being applied to the binarized image. The outline and centroid of the ball are displayed on the image

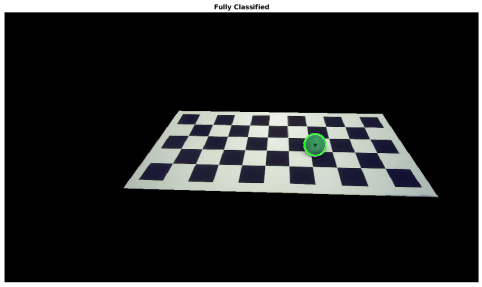

Fig. 15. Circle overlay onto initial image of the ball. The color of the circle matches the color of the ball that was detected

Fig. 16. Image of robot arm with balls on grid before object recognition occurs

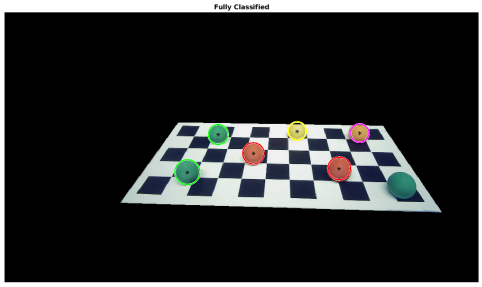

Fig. 17. Cropped checkerboard image with center points for each ball after full detection sequence

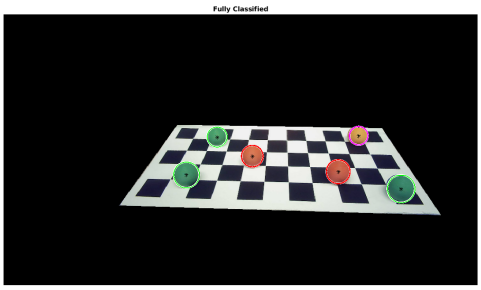

Fig. 18. Cropped image after the first ball was taken with center points for each remaining ball

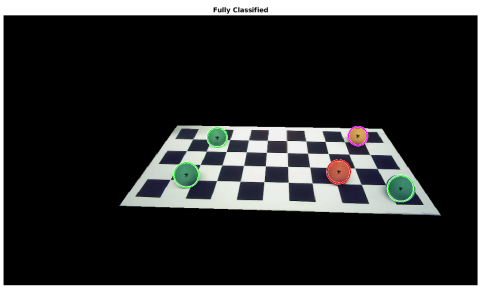

Fig. 19. Cropped image after the second ball was taken with center points for each remaining ball

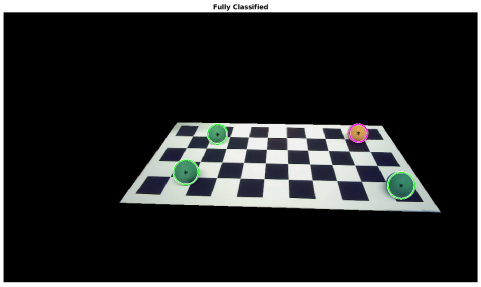

Fig. 20. Cropped image after the third ball was taken with center points for each remaining ball

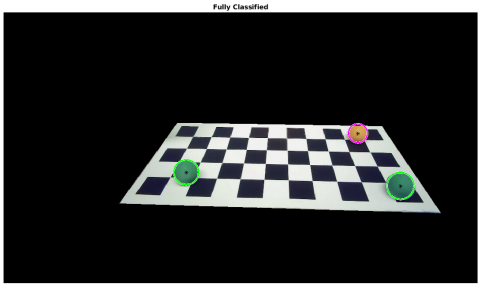

Fig. 21. Cropped image after the fourth ball was taken with center points for each remaining ball

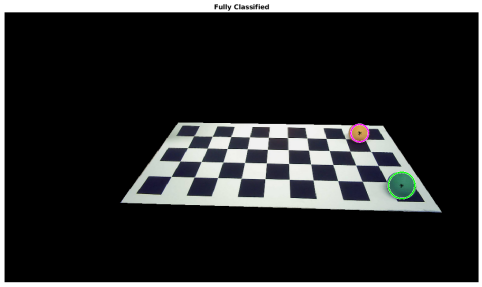

Fig. 22. Cropped image after the fifth ball was taken with center points for each remaining ball

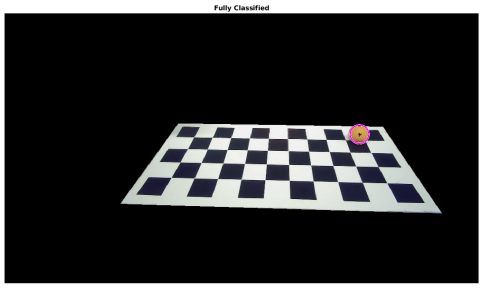

Fig. 23. Cropped image after the sixth ball was taken with the center point of the last ball

IV. DISCUSSION

A. System Architecture

The code was split into Robot.m, Camera.m and labFinal.m to keep the overall structure simple and clear to those reading the code. All the functions that involved motion were within the Robot class. The Camera class was used for our image recognition, object identification, and robot frame conversion. Within labFinal.m, only functions from the Robot and Camera classes are used. Initializing the camera within labFinal.m is just one line, but within Camera.m many other functions are run. This includes calibration of the camera to remove the fish-eye effect of the lens, calculating the position of the camera, and initializing the grid points. These initialization functions are all done in the background, which makes the code much neater through compartmentalization. Within the sorting process, getBalls() is a function that utilizes other functions within Camera, but again these are compartmentalized to be in the background. As a result of the compartmentalization of code, particularly camera code, the labFinal.m file that runs the sorting sequence is focused and reads linearly. This makes understanding and debugging the surface-level code much easier.

B. Color Detection

The team experimented with several combinations of different filters and processing techniques before choosing the pipeline described in the Methods section. The median filter was consistent in removing noise and did not significantly impact the integrity of the ball's shape. Erosion and dilation together helped remove noise around the ball and fill in some of the circle shape that was lost during the color filtration. Binarizing the image allowed the team to use the pre-made MATLAB function imfindcircles() to detect the centroid of the ball.

When choosing colors to use for the final demonstration, the team elected to not use black, gray, or white balls. Black and white balls cannot be used in the team's setup because black and white squares are used to denote the checkerboard. This means during calibration of the color detection, the balls would blend in with the checkerboard, making them impossible to detect in a way that is separable from the area around them. In testing, we had a similar issue using gray balls, as the shadows cast on the checkerboard by the lighting could create gray areas in the image. These gray areas were then mistakenly recognized as gray balls.

C. Overall Performance

Within this lab there were three major components to our code: object detection, camera to robot frame conversion, and robot control.

The object detection with the camera had very consistent performance, as it was able to find all balls within frame very consistently. This is due to the careful calibration and image processing, which made it both responsive, for live tracking, and reliable for objects beyond the balls: the team was also able to use the camera for detecting a rubber duck and a knife. Though the object detection is not always perfect, as seen in Figure 17, which has three green balls within the frame, but only two are recognized. This error is mitigated by re-detecting the field after each motion in the event a ball has been added or moved. This can be seen in Figure 18 where the previously unrecognized ball is eventually detected.

The camera to robot frame conversion is fairly reliable, but is prone to slight error that is largely mitigated by the size of the gripper. The first of the two parts of the conversion is the camera to checkerboard frame conversion: this is where the majority of the error comes from. This error is due to the lack of precision when measuring from the camera to the checkerboard frame. For example, the team determined that the camera is not exactly center to the board, but rather has a 3mm offset. While the team was able to adjust for the majority of the error, there is residual error. There is also some error caused by the adjustment from the camera's coordinates for the balls to the true coordinates of the balls. The other part of the conversion, checkerboard to robot frame, is more accurate due to better measurements and the general inability to move the checkerboard and robot base. In comparison, the camera can be rotated, which can alter the height and position of the lens.

Overall, the robot is able to go to any spot on the checkerboard with both a high degree of accuracy and precision, but it tends to be slightly off of the location of a ball due to the previously mentioned error. In practice this is largely irrelevant due to the large size of the gripper.

The final component of the project is the robot control, which has been extremely reliable. The team has a high degree of control over the robot with its functions, such as

the approach function. This makes it very easy to manipulate for this use-case. In this lab, the team primarily utilized interpolated motion, as it was both smooth and responsive throughout the various applications, from ball sorting to live tracking.

V. CONCLUSION

In this lab, the team synthesized skills learned in previous labs and new computer vision skills to program the OpenManipulator-X arm to sort colored balls. The team revisited forward kinematics, inverse kinematics, and trajectory planning to control the robot throughout the challenge. Additionally, the team explored image processing techniques for color detection and mathematics to navigate seamlessly from the camera frame to the robot's task space. This exercise serves as a foundation for potential future work in the robotics field working with robotic arms or computer vision systems.

APPENDIX

A. Appendix A: Links to Lab Code and Demonstration Video

Link to code on GitHub: `https://github.com/RBE3001-A23/RBE3001_A23_Team19/releases/tag/lab-5-final`

Link to YouTube video of demonstration: `https://youtu.be/fyROhRms9YE`

B. Appendix B: Table of Contribution

	Ashe	Sebastian	Jakub
Architecture	33%	33%	34%
Coding	33%	33%	34%
Experimentation	33%	34%	33%
Documentation	60%	15%	15%
Video creation	15%	15%	60%
Result analysis	34%	33%	33%

REFERENCES

[1] "Understanding what is a robot vision system," Techman Robot, https://www.tm-robot.com/en/robot-vision-system/.