RBE 3001: Automated Object Sorting with a 4-DOF Arm Robot

Kevin Siegall, Robert Gunduz, and Melissa Kelly

Abstract— **This lab report is the culmination of all of our work in this class. We created a system that picks up and sorts objects by color. In order to accomplish this task, computer vision was implemented to recognize, locate, move to, pick up, and sort the different objects. Since there were no specific instructions on the approach for the different methods in this lab, there was some trial and error when testing to see what approach worked best for the robot.**

I. INTRODUCTION

The purpose of Lab 5 was to utilize basic joint control, forward kinematics, inverse kinematics, velocity kinematics, and trajectory planning to create a robotic system to pick up and sort various objects. This system would include recognition and location of the objects, and move to, pick up, and then sort the different objects by color. The first step in this lab was setting up the given camera and calibrating it to set up the workspace. This calibration included fixing the fish-eye-lens distortion and performing both intrinsic and extrinsic calibration. Then, image processing was used to implement the classification and detection of the centroid of solid-colored, spherical objects. Once the detection of the object was completed, color masks were created to differentiate the objects by color, with the intent of later sorting the objects based on their color. The color masks were created using HSV values. When localizing each of the objects in the workspace, it was important to have the proper height of the camera. The centroid and radius of the objects were used to assure that the camera was not recognizing the objects as flat in the workspace and so that the robot would try and pick up the object at the right height. For the final challenge portion of the lab, a MATLAB script was created that determined the object's position, uses inverse kinematics to calculate the required joint angles of the end effector, uses trajectory planning for a smooth motion, closed the robot gripper, picked up the object, moved it to a different position then where it was picked up, and dropped the ball off at the new position. In addition to the challenge, real-time object tracking was implemented so that the robot would be able to follow an object around the workspace until the object left the workspace and was no longer detected. Although the lab called for the use of spherical objects, other objects were used in this lab such as a rubber duck and colored dice.

II. METHODOLOGY

A. Basic Joint Control

In the first lab, we wrote five methods to be used in later assignments. servo_jp, a method to move the arm to a given position. Following, we constructed interpolate_jp, which would achieve the same goals of the previous method, however it would use interpolation instead to allow for more control. Next, we constructed measured_js, which would display position and velocity data of the arm joints when requested, which we could use for debugging purposes. The fourth function setpoint_js was designed to show the intermediate goal points the servos were using at any given time. Finally, goal_js would allow us to see the final goal point for the servo arms at the current moment.

We Tested servo_jp and interpolate_jp against each other by moving the robot through a set of waypoints three times each. the robot was then made to move to four arbitrary poses, again one with and one without interpolation. The results were then compared to measure the effectiveness of each approach. Moving forward we opted to use interpolate_jp extensively for more control over basic joint motion within more complex functions.

B. Forward Kinematics

We calculated the Forward Kinematics of the 4 DOF Robot by using the Denavit-Hartenberg (DH) convention. We implemented a table of the DH parameters on MATLAB from the base frame of the robot to the end effector over four joints. We created a method to find the symbolic 4x4 homogeneous transformation matrix for each row called dh2mat().

Once all the transformation matrices were constructed, they are fed into dh2fk(), a method we made to represent the transformation from the robot's base frame to the end effector frame by multiplying all the intermediate frames together. Using the symbolic transformation matrix, we could read current joint angles to solve for current position and orientation of the robot's end effector, which we achieved in the fk3001() method.

We constructed a 3-D stick plot of the Robot arm using the prevoius methods. Thus allowing us to monitor each frame's pose in real time. To provide more information for debugging purposes we wrote three methods: setpoint_cp(), goal_cp(), and measured_cp(). These three methods fulfil the same function as their counterparts from Basic Joint control, however display a homogeneous transformation matrix from the base frame to the end effector.

Our experimentation entailed moving the robot away from the home position and back multipole times for error checking. We coded a plot_arm() method to facilitate the live 3 D stick model of the arm. We included the representation of unit vectors of each frame to show orientation. We then wrote MATLAB script for the robot to move along a triangular path to observe the live plot.

C. Inverse Kinematics

Our third objective was to develop trajectory planning via inverse kinematics of the 4 DOF Robot. We derived equations correlating end effector position and orientation with the four joint parameters. The inverse kinematic solution would be fed to the robot to execute movements in task space.

During testing, the Robot was made to trace a triangular path along the task space of the robot. During this testing we recorded joint space and task space transformations. We wrote the method ik3001() to help command the robot to move to different positions in task space given a desired end effector position.

We implemented cubic and quintic trajectory planning methods to further control robot movements. The cubic_traj() method plans trajectory given start and ending velocities position and time. The second method, quintic_traj(), utilizes start and ending accelerations in conjunction with prior parameters.

To test trajectory planning we created the run_trajectory() function, to combine the inverse kinematic method and trajectory planning methods. Both cubic and quintic trajectories were tested by passing three poses for the robot to loop through. The series of tests run with quintic trajectories were placed into a separate method called quintic_traj(), which we used going into the final project for trajectory planning.

D. Velocity Kinematics

In this lab, we implemented velocity kinematics for a robot arm by calculating the 6x4 manipulator Jacobian matrix. We used the partial derivative approach to find the upper half (3x3) of the Jacobian matrix and created a MATLAB method called 'jacob3001(q)' to calculate and validate it. To test the Jacobian matrix, we input a 4x1 vector to move the robot arm to a singularity configuration, to check the first column of the Jacobian and that its determinant was close to zero. For forward velocity kinematics, we used the 'fdk3001()' function, which solved the $\mathbf{p} = J(q)\mathbf{q}$ equation in real-time, taking joint angles and instantaneous velocities as inputs to return task-space linear and angular velocities. Real-time visualization was achieved by plotting a stick model of the arm following a quintic trajectory drawn by moving the robot to specific triangle vertices. We also implemented an emergency stop and safety check by recording the determinants of the 3x3 Jacobian submatrix. If the determinant values approached zero or became too close, meaning the arm was approaching a singularity, the code displayed an error message on the live plot and stopped the robot's motion.

E. Final Project

In this lab, we combined everything we learned and already implemented this term to create an automatic robotic picking up and sorting system. To complete this task we first had to set up and calibrate the camera (using intrinsic calibration) to detect and define the field we would be working in. The next step was to register the camera-to-robot connection using extrinsic calibration. To register the connection between the camera and robot, we had to register the reference frame of the robot and the reference frame of the generated image from the camera, which was used to relate the position of the objects and the task space coordinates. For the object detection and classification section, image processing was used to determine the image processing pipeline that uses the image to find the centroid of the spherical object (using the radius) and then draws a circle of the color of the object on the camera feed. We implemented color recognition by using the HSV color space and created color masks for each of the colors we used (red, green, yellow, orange, and purple). In order for the robot to pick up the ball, we had to convert the centroid location of the objects into target positions that the robot can use. We also had to use the pointsToWorld() function so that the objects didn't project as flat objects in the workspace.

For the final project challenge itself, we created a MATLAB script that determined the 3D position of an object with respect to the robot's reference frame, used inverse kinematics from previous labs to calculate the joint angles required for the robot's end effector to reach the object's location, Use trajectory planning to implement a smooth motion from the robot's current position to the location of the object by moving the robot to a position above the object and then lowered straight down, then close the robot gripper, picked up the object, moved it to a different position then where it was picked up, and dropped the ball off at the new position. The robot then sorted the objects by color to arbitrary positions. The system would run until there were no more objects left to detect in the workspace. We also completed the implementation of real-time object tracking, where the robot would be able to follow an object that was being moved around the workspace, and would run until the object was no longer detected in the workspace. Lastly, we added a purple color mask so that the robot can detect and sort purple dice as our choice object (the robot was also able to detect and sort a yellow rubber duck).

TABLE I
DH PARAMETER TABLE

	θ	d	a	α
T_1^2	θ_1	$L_0 + L_1$	0	-90
T_2^3	$\theta_2 - \phi$	0	L_2	0
T_3^4	$\theta_3 + \phi$	0	L_3	0
T_4^{ee}	θ_4	0	L_4	0

III. RESULTS

The first thing we needed to derive for our robot arm was the Forward Kinematics solution. We did this using the Denavit-Hartenberg (DH) Method, as shown below.

Now that we have our DH Table, deriving our final Forward Kinematics solution is pretty simple. For a given row of the DH Table, the transformation matrix can be found

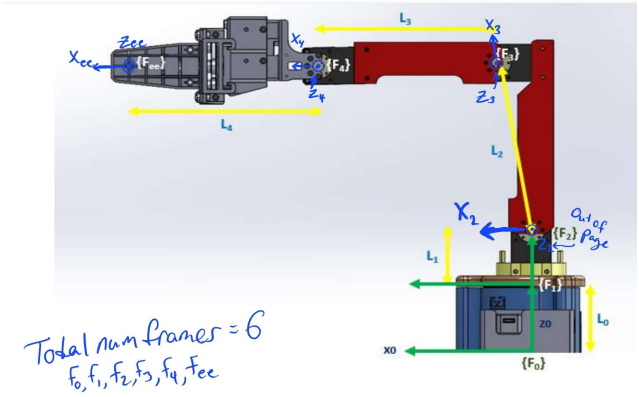

Fig. 1. The DH Frames for the Robotic Arm

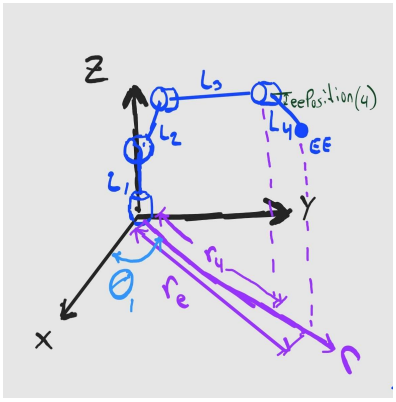

Fig. 2. 3D Drawing of the robotic arm

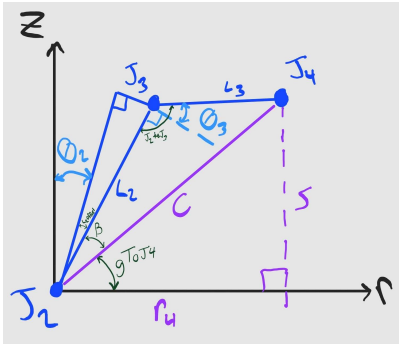

Fig. 3. 2D Drawing of the robotic arm

like this:

$$T_i^{i+1} = \begin{bmatrix} cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i sin\alpha_i & acos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & asin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Once we have that, getting the final transformation matrix for our arm was as simple as multiplying them all together.

$$T_0^{ee} = \prod_{i=0}^{4} T_i^{i+1} \tag{2}$$

In the end, we were left with this:

$$R_0^{ee} = \begin{bmatrix} \frac{c_{1234}+c_{234-1}}{2} & -\frac{s_{1234}+s_{234-1}}{2} & -s_1 \\ \frac{s_{1234}-s_{234-1}}{2} & \frac{c_{1234}-c_{234-1}}{2} & c_1 \\ -s_{234} & -c_{234} & 0 \end{bmatrix}$$

$$\mathrm{P}_0^{ee} = \begin{bmatrix} (L_2 * c_{\phi 1-2} + L_2 * c_{12-\phi} + L_3 * c_{23-1} + \\ \quad L_4 * c_{1234} + L_4 * c_{234-1} + L_3 * c_{123})/2 \\ (L_2 * s_{\phi 1-2} + L_2 * s_{12-\phi} - L_3 * s_{23-1} + \\ \quad L_4 * s_{1234} - L_4 * s_{234-1} + L_3 * s_{123})/2 \\ L_0 + L_1 - L_3 * s_{23} + L_2 * s_{\phi-2} - L_4 * s_{234} \end{bmatrix} \tag{3}$$

The next thing we had to do was start working on our Inverse Kinematics. We used the geometric approach for deriving it, and our diagrams can be seen below.

Using the diagrams we derived the following equations for Geometric inverse kinematics. Starting with deriving for θ_1 with the following equations from the diagrams:

$$r_e = \sqrt{eePosition(1)^2 + eePosition(2)^2} \tag{4}$$

$$\cos\theta_1 = eePosition(1)/R_e \tag{5}$$

$$r_4 = r_e - L4\cos eePosition(4) \tag{6}$$

$$\theta_1 = atan2(\pm\sqrt{1-\cos\theta_1{}^2}, \cos\theta_1) \tag{7}$$

The angle θ_1 can be derived independently from the other three joint angles, as seen in Figure 2. The Next series of equations are used to solve for the two linked angles, θ_2 and θ_3. Below are the calculations relevant for θ_2:

$$s = r_e + L_4 \sin eePosition(4) - L_1 \tag{8}$$

$$C = \sqrt{r_4^2 + s^2} \tag{9}$$

$$L_2Offset = atan2(24, 128) \tag{10}$$

$$\theta_2 = 90 - gToT_4 - \beta - L_2Offset \tag{11}$$

Below are the relevant calculations for θ_3:

$$\cos gToJ_4 = r_4/s \tag{12}$$

$$gToJ_4 = atan2(\pm\sqrt{1-\cos gToJ_4{}^2}, \cos gToJ_4) \tag{13}$$

$$\cos\beta = \frac{L_2^2 + C^2 - L_3^2}{2L_2C} \tag{14}$$

$$\beta = atan2(\pm\sqrt{1-\cos\beta^2}, \cos\beta) \tag{15}$$

$$\cos J_2toJ_4 = \frac{L_2^2 - C^2 + L_3^2}{2L_2L_3} \tag{16}$$

$$J_2toT_4 = atan2(\pm\sqrt{1-\cos J_2toT_4{}^2}, \cos J_2toT_4) \tag{17}$$

$$\theta_3 = 90 - J_2toJ_3 + \phi \tag{18}$$

From Figure 3, the mathematical relationsips for both of these angles are derived. Once θ_2 and θ_3 are known, the final

angle θ_4 is found as the difference of the previous two joint angles and the desired end effector orientation:

$$\theta_4 = eePosition(4) - \theta_2 - \theta_3 \tag{19}$$

We calculated the Jacobian matrix by using the time derivative method. where we took the translation component of the full transformation matrix and derivated with respect to each joint variable. Below are the translation column of the final transformation matrix and the column wise results of the derivation:

$$P_0^{ee} = \begin{bmatrix} C_1 * 620C_{23} + \frac{667}{5}C_{234} + 8 * \sqrt{265}\sin(\theta_2 + \phi) \\ S_1 * 620C_{23} + \frac{667}{5}C_{234} + 8 * \sqrt{265}\sin(\theta_2 + \phi) \\ 96.326 - \frac{667}{5}S_{234} - 8 * \sqrt{265}\sin(\theta_2 + phi) - 124S_{23} \end{bmatrix} \tag{20}$$

$$J_{p1} = \begin{bmatrix} -S_1 * 620C_{23} + \frac{667}{5}C_{234} + 8\sqrt{265}\sin(\theta_2 + \phi) \\ C_1 * 620C_{23} + \frac{667}{5}C_{234} + 8\sqrt{265}\sin(\theta_2 + \phi) \\ 0 \end{bmatrix} \tag{21}$$

$$J_{p2} = \begin{bmatrix} -C_1 * 620S_{23} + 667C_{234} - 8\sqrt{265}\cos(\theta_2 + \phi) \\ -S_1 * 620S_{23} + 667C_{234} + 8\sqrt{265}\sin(\theta_2 + \phi) \\ -124C_{23} - \frac{667}{5}C_{234} - 8\sqrt{265}\cos(\theta_2 + \phi) \end{bmatrix} \tag{22}$$

$$J_{p3} = \begin{bmatrix} -C_1 * 620S_{23} + \frac{667}{5}S_{234} \\ -S_1 * 620S_{23} + \frac{667}{5}S_{234} \\ 124C_{23} - \frac{667}{5}C_{234} \end{bmatrix} \tag{23}$$

$$J_{p4} = \begin{bmatrix} -\frac{667}{5}C_1S_{234} \\ -\frac{667}{5}S_1S_{234} \\ -\frac{667}{5}C_{234} \end{bmatrix} \tag{24}$$

The bottom part of the Jacobian can be determined from the z rotation component of the Transformation matrix T_0^i, with i being the frame of the joint being solved for. The rotation component of the Jacobean for our robot is the following:

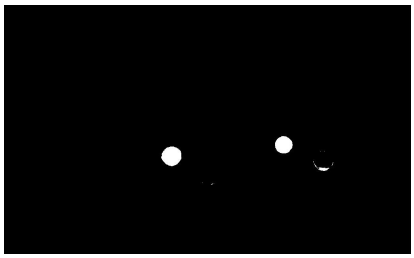

Fig. 4. Image after red mask is applied

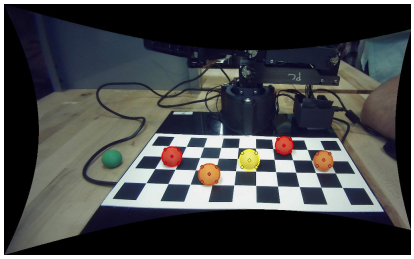

Fig. 5. Image showing proper color recognition

$$J_o = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \tag{25}$$

Now that we have our derivations, we could get on with the actual requirements for the final project. After calibrating, our next step was to create and apply our color masks. Using Matlab's ColorThresholder tool, we created masks in the HSV color space. With those, we were able to isolate colors into black and white images (Fig 4) and use the regionprops function to extract centroid and diameter data for objects that passed the mask. We then draw the centroid and circle onto the original image for later reference (Fig 5)

From there, we ran a quick check to make sure we didn't continue processing data for objects with diameters too small to be our targets. We assume than any objects we are still seeing are one of our ball targets, and convert the pixel centroid position into real world coordinates, and then adjust the exact position to account for the angle projection from the camera (Fig 6)

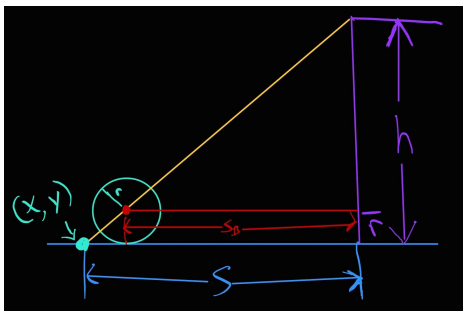

Fig. 6. Side view of camera looking down at the ball

Fig. 7. Arm in snapshot position

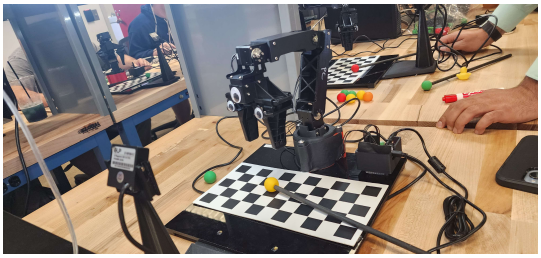

Fig. 8. Robot arm during live tracking

$$S_B = \frac{S * (h - r)}{h}$$

$$\theta_{C->B} = atan2(Y_C - Y_B, X_C - X_B)$$

$$(X_B, Y_B) = (X_C - S_B * cos\theta_{C->B}, S_B * sin\theta_{C->B}) \quad (26)$$

Once we adjust our ball positions, we then do a final check to make sure that the ball is in a place we can grab it. In our case, this took the form of us taking the known world coordinates of the corners of our field, and removing any targets whose centroids were outside of our checkerboard field.

That is all of our theoretical math, so the next step was to figure out what our exact procedure would be for the final project. What we decided on is as follows:

1) Move the arm out of the way, to some predefined location (Fig 7)
2) Take a picture with the camera, mask, and locate targets
3) Choose a random target of a random color. If no balls are found, return to step 1.
4) Use quintic interpolation in joint space to move the end effector above the ball
5) Lower the end effector onto the ball using quintic interpolation
6) Close the gripper
7) Raise the end effector with the newly collected ball
8) Use quintic interpolation in joint space to move the arm to a predetermined position based on the target's color
9) Lower the arm, and open the gripper to release the ball
10) Repeat until program is exited.

For live tracking, our procedure was only modified a little:

1) Take a picture with the camera, mask, and locate targets
2) Choose a valid target to follow
3) Use interpolate_jp to move above the target
4) Repeat

IV. DISCUSSION

A. Calibration and Image Processing

Through the course of programming the 4 DOF arm robot to link up to the camera we needed to run calibrations for the camera used. The fish eye lens distorts the images taken with the camera, so the distortion is corrected using the built in camera calibrator from the Matlab extension for image processing tools. We chose a position to move the robot out of the way of the camera to let it calibrate upon startup.

When finding calibration parameters for the camera, we needed to take multiple pictures of the workspace from the camera in different orientations. As part of this process, we pruned a sizeable collection of these images, such that the image re-projection error was less than 1 pixel. Beyond this, we had a few errors in our initial calibrations, warranting further modifications to the code. Primarily, the calibrator detected another two columns of checkerboard vertices on either side of the real board, which needed to be filtered out.

Using the image processing tools, the images can be masked to isolate a specific groupings of pixels by color values. Our team used the HSV color model to better account for changes in the light conditions of the lab. The color space proved to be very robust under most conditions. However had trouble discerning certain colors from certain lighting conditions, such as blue and purple, which we used for the additional random objects (Subsection D). When these colors are searched for, the program sometimes mistakenly recognizes shadows over the black checkerboard pattern as objects.

The masked image is then used to identify the centroid of the target object, as well as it's approximate radius in pixels. To avoid mistaking small groupings of pixels as an object, the dimensions of the potential object are filtered based on a minimum radius. Through the use of forward/inverse kinematics, the base frame of the robot is linked to the base frame of its workspace and the position of the camera.

To account for the offset of the raw camera object locations, mathematics using similar triangles is applied to determine the actual position from the object's radius and centroid position. When testing the code to move the gripper to the arm, the raw value caused enough overshoot to have the end effector miss the target entirely.

B. Automated Object Sorting

We utilized a while loop to repeat the sequences of methods used throughout the process. Joint space interpolation was implemented for our arm instead of task space interpolation since the movements of the later method were not as fluid as we would have desired. We also utilized quintic trajectory planning to allow for more control over arm movements and remove the abrupt stops from the raw

interpolation. using interpolation only could cause the object to move out of the way accidentally before the end effector can grabbing it. The color of the object to be removed is chosen at random. We programmed color recognition for green, red, orange, yellow, and gray, although not all of these colors are required. Blue and purple filters were also generated, but are not used due to the detection errors found in initial image processing.

Once our gripper is in position, we would lower the open gripper gently to the target object and call a close gripper method provided by the given lab code. The arm would then raise and move to a designated dropping zone for each of the colored objects off to the side of the workspace. To prevent our robot from recognizing objects already outside of the workspace, the program would additionally filter out any objects sensed outside of the workspace, defined by its corners in pixel coordinates of the image.

After the ball is dropped into the designated dumb zone, which we predefined in a matrix of task space coordinates, the loop would repeat until no objects are detected on the workspace. Since the loop randomly selects a color to scan for, it is possible for there to be a loop in our code where the Robot does not sense an object on the board, thus our program loops until it picks a color of which there is an object to be removed from the workspace. As part of the loop, we have the robot arm return to the calibration position before moving to the next object in the loop.

C. Live tracking

Afterwards, to implementation the live tracking of a moving object, we constructed a separate loop, with a simplified work load, in which the gripper maintains a set distance above an object, which we set to 100mm. once the arm has completed this movement, the program loops and repeats the process. We utilized raw interpolation to make these movements instead of quintic trajectory planning. Due to the non-blocking nature of raw interpolation, our robot arm can make quick course corrections mid trajectory. In this regard, we prioritized the reaction time of our robot over the fluidity of its movements.

We determined it would be problematic if the robot randomly choose which color to scan for each loop, while also desiring a robust code that can work under multiple conditions. Hence, the first color recognized, of the list we prepared, which is present on the board is the object the robot will choose to follow.

D. Additional Random Objects

To account for additional random objects, our code needed only slight modifications to recognize and sort objects out of the scope of the initial parameters of the project. We noticed some objects we placed on the board were large enough to be recognized as objects, but too small to be grabbed by the gripper. Conversely, we also placed an object which did not allow the gripper to close completely, which we believe caused an error in our Open gripper method.

TABLE II
APPENDIX: FINAL PROJECT CONTRIBUTIONS

Section	Members Contributed
Planning	Kevin Siegall, Melissa Kelly
Coding	Kevin Siegall
Experimenting	Kevin Siegall, Melissa Kelly
Analyzing Results	All
Write Up	All
Creating Video	None

Paper	Members Contributed
Abstract	Melissa Kelly
Introduction	Melissa Kelly
Methodology	Melissa Kelly, Robert Gunduz
Derivations	Kevin Siegall
Results	Kevin Siegall
Discussion	Robert Gunduz
Conclusion	Robert Gunduz

Excluding edge cases, depending on the color of the random object placed on the board, the Automated Object sorting loop might be able to remove the object without modification. Due to the abstractions we made in our calculations beforehand, we were able to account for varying object image offsets.

While testing, we used purple objects to test alternative HSV filters. While many trials ended with successful object sorting, there were still minor instances of the filter detecting non existent objects.

E. CONCLUSION

Using the methods we built from previous labs, we successfully managed to make the robot execute a complex series of task. Camera calibration was utilized to properly calibrate the mounted camera to the given workspace. Image processing allowed us to identify the objects on the task space through masking the object's color. The various movement methods we constructed up to this point moved the arm across the work space in deliberate manner. Our robot is capable of clearing and sorting the objects from the workspace. Furthermore, the robot was able to closely follow a moving object across the workspace and remove odd objects. With this, we have demonstrated a comprehensive understanding of the course material, which we will use utilize greatly in our careers moving forward.

APPENDIX A: GITHUB RELEASE LINK

This term we worked using Github for version control. Our final code release can be found here, or using the link below:

`https://github.com/RBE3001-A23/RBE3001_A23_Team18/releases/tag/Lab_5`

ACKNOWLEDGMENT

Praise the SAs, they are the best.
The graders too (hi Kalina!)