Lab 5: Final Project

Aiden Deady, Aman Hiregoudar, and Aliaa Hussein

Abstract— **This paper discusses the implementation of a robotic pick and place system. Calculations and execution of forward kinematics, inverse kinematics, trajectory generation and differential kinematics are discussed. Additionally, the incorporation of computer vision to locate the target objects is described along with associated diagrams of the image processing pipeline. This paper also provides an overview of the design of the system architecture, workflow and state machine.**

I. Introduction

The objective of this project is to develop a robotic pick and place sorting system using a 4 degree of freedom robotic arm and MATLAB. For the system, forward and inverse kinematics for position and velocity and trajectory generation which was completed in previous labs is used. Additionally, computer vision is incorporated into the system to locate target objects and localize them with respect to the robot. This then allows the robot to move towards the object, pick it up and place it in a specified location using the grippers.

For this project, different colored balls are placed in the workspace of the robot and are then picked and sorted by color (red, orange, green, yellow). An image of the workspace is taken to identify the balls and the position is then determined with respect to the robot's reference frame using intrinsic and extrinsic calibration. Inverse kinematics is then used to determine the joint angles required for the robot's end effector to reach the ball. For the robot to smoothly move to the location of the ball, trajectory planning is used. The robot initially moves slightly above the location of the ball with the grippers open, and then straight down to avoid hitting the ball. Then robot's grippers are then closed to pick the ball. Based on the given ball's color, it is then placed on the outside corners of the workplace, which is the selected location for sorting, by opening the grippers. The robot will continuously pick and place the balls until there are none in the workspace.

II. Methods

A. Camera Setup and Intriscic Calibration

To set up the camera, ensure that there is a connection between the robot and the camera by viewing the live video feed of the camera. The robot's workspace was in an environment with bright lighting to ensure that the colors and intensity of the target objects from the camera's view remains consistent and does not affect the color segmentation. For intrinsic calibration of the camera, the 'Camera Calibrator' app on Matlab is used to take images of the checkerboard using the camera, where the dimension of the square is 25 mm. A total of 40 images are taken at various angles around it to maximize the correction of the distortion of the camera optics. The images are then reviewed and removed based on the alignment of the x and y axes, location of the origin (0,0) of the checkerboard, and the number of detected points of the cross section of the grid on the checkerboard. The 'fisheye' camera model is selected, and camera calibration is run. An analysis of the reprojection errors is done by checking the mean and discarding any image with an error greater than 1 pixel. The camera parameters from the calibration are then exported and saved as a script to be used for creating a camera object.

B. Camera-Robot Registration

For extrinsic calibration, registration between the reference frames of the robot and image is completed to determine the robot task space coordinates from its relation to the object in the camera's view. The transformation matrix from the checkerboard to image frame is determined using the **getCameraPose()** function. The transformation of matrix from the base frame to the checkerboard frame is also calculated. Both transformation matrices are used to calculate the transformation from the x-y pixels in the image plane to the 3D position in the robot's frame.

The **getCameraPose()** function is run to get the camera's extrinsic calibration parameters. The data tips tool is then used to determine the x-y coordinates in pixels in the image frame for a given point. The **pointsToWorld()** function is then used to transform from the image frame to the checkerboard frame, outputting the coordinate in mm with respect to the reference frame of the checkerboard. The transformation matrix from the base frame to the checkerboard is then multiplied by the pose in the checkerboard frame to calculate the target object's position in the robot's base frame.

To ensure that this robot-camera calibration registration is correct, an image of the task space is taken using the **cam.getImage()** function and displaying is using the **imshow()** function. Four points that are known in the robot's reference frame are selected and identified using the data point tool. These four pixel coordinates are converted to coordinates in the checkerboard's frame and then coordinates in the robot's frame using the transformation matrix from the base frame to the checkerboard frame. The resultant coordinates are then checked in correspondence with expected values, which can be easily determined by sending the robot to the given coordinates and observing whether it aligns with the four initially selected points.

C. Object Detection and Classification

Image processing methods are used to determine the centroid of a solid-colored spherical object and place a marker on the camera feed. Initially, an image of the workspace is taken, and a mask of anything outside of the checkerboard is placed using the coordinates of the checkerboard to ensure no target objects that are outside the workplace are detected. A color mask is then created for each of the colors of the target objects (red, yellow, orange and green) using an HSV filter and the function for each corresponding color is exported. The binary images of each of the color masks are combined using bitwise 'or'. The combined masked image is then eroded using the **imerode()** function to eliminate any small number of pixels that could affect the identification of the centroids of the colored balls. The masked binary image is dilatated using the **imdilate()** function to increase the size of the mask. Additionally, the **imfill()** function is used to fill any holes in the outline of the colored balls to ensure the centroid is calculated accurately.

For the calculation of the centroids, the final processed image is used as an input for the **regionprops()** function. The centroids are then displayed and labeled based on the color of the mask for each corresponding centroid.

D. *Object Localization*

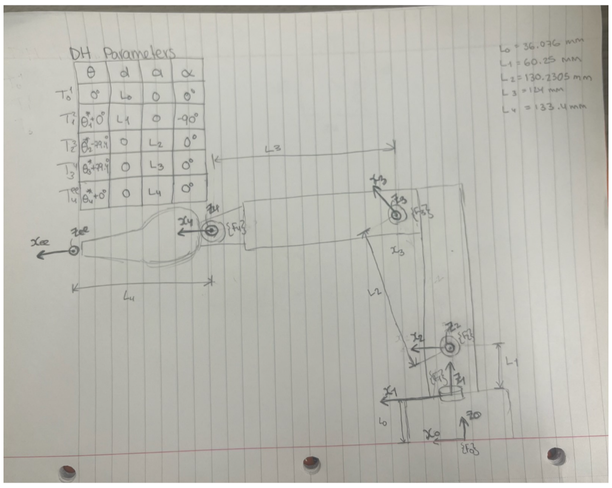
One of the problems we had to solve regarding the position of the ball was the offset between the calculated position of the ball and the actual position. The camera sees the ball at an angle as a flat circle, and therefore produces a position point behind where the ball actually is. If this original position is used, the robot would navigate to a position behind the ball instead of above the ball.
In order to solve this problem, we used trigonometry and similar triangles to determine the offset. We can calculate the horizontal distance from the camera base to the point that the camera thinks the ball is. We also know the height of the camera from the surface, and the radius of the ball. Using these measurements, we can form two similar triangles, and therefore find the horizontal distance from the ball to the camera.
At the same time we can find the angle of the ball with respect to the camera base using the projected point. This angle is the same for the real ball position, and therefore can be used along with the real distance to the ball to find the exact position of the ball. This position can then be fed into the transformation matrices instead of the projected point to get the real position of the ball with respect to the robot base.

E. *Design of Pick and Place Sorting System*

As stated above, the final project goal was to implement all of these ideas to develop a sorting pick and place ball system that would run continuously until there were no balls left in the workspace.
The first step of this is to calibrate the camera. From here a while loop was implemented that would run as long as there were centroids detected in the workspace by the camera. The while loop first gets the image and creates a mask of the checkerboard so that nothing outside of the board is detected. The next step is to use the color masks in conjunction with the checkerboard mask so that we can accurately determine the colors of the balls. To make sure the image is clean, the image is run through the erosion, dilatation and fill functions generated by MATLAB.
Now that the image is clean, the next step is to find and plot the centroids. One issue with the final project is what to do when there are no centroids. Our group decided to use robot.interpolate() and hard code a specific space for the robot to go to if there are no balls in the workspace by implementing a try and catch in the code.
After, the transformation calculations are found using cam.getCameraIntrinsics, cam.ConvertCam2RealPos and doing inverse kinematics to get the values of the joints that the robot needs to get the ball. This is where we accounted for the error of what the camera sees when it comes to the balls position and the real position in reference to the robot arm.
The final step is putting all of this together with trajectories to the ball which includes picking up and dropping off. We decided to use a cubic trajectory for this task. By implementing if statements we hard coded a specific drop off point for the robot to travel to based on the color of the ball.

III. RESULTS AND DISCUSSION

A. *Forward and Inverse Position Kinematics*



DH Parameters

	θ	d	a	α
T_0^1	$0°$	L_0	0	$0°$
T_1^2	$\theta_1^*+0°$	L_1	0	$-90°$
T_2^3	$\theta_2^*-79.4°$	0	L_2	$0°$
T_3^4	$\theta_3^*+79.4°$	0	L_3	$0°$
T_4^{ee}	$\theta_4^*+0°$	0	L_4	$0°$

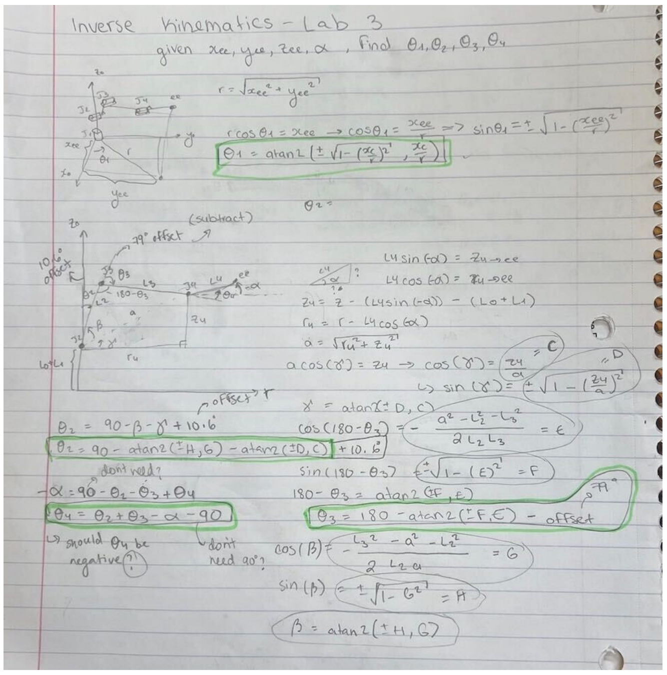
Figure 1: DH Parameter of the Robot Arm

Fig. 1 displays the robot arm home configuration with each joint's respective frame as well as the DH parameters for each joint. We calculated the frames and DH parameter by following the DH convention. The parameters are then used for the forward kinematic transformation matrix from the robot's base frame to the end effector.

1	0	0	281.35
0	0	1	0
0	-1	0	224.33
0	0	0	1

Figure 2: Transformation Matrix of the Home Position (0,0,0,0)

Fig. 2 displays the transformation matrix when the robot's angles are at (0,0,0,0) using forward kinematics. The values were determined using the **fk3001()** function.



Inverse Kinematics - Lab 3

given $x_{ee}, y_{ee}, z_{ee}, \alpha$, Find $\theta_1, \theta_2, \theta_3, \theta_4$

$r = \sqrt{x_{ee}^2 + y_{ee}^2}$

$r\cos\theta_1 = x_{ee} \rightarrow \cos\theta_1 = \frac{x_{ee}}{r} \Rightarrow \sin\theta_1 = \pm\sqrt{1 - \left(\frac{x_{ee}}{r}\right)^2}$

$\boxed{\theta_1 = \text{atan2}\left(\pm\sqrt{1-\left(\frac{x_c}{r}\right)^2}, \frac{x_c}{r}\right)}$

$\theta_2 =$

(subtract)

79° offset

10.6° offset

$L4\sin(-\alpha) = z_{4 \rightarrow ee}$

$L4\cos(-\alpha) = r_{4 \rightarrow ee}$

$z_4 = z - (L4\sin(-\alpha)) - (L_0 + L_1)$

$r_4 = r - L4\cos(-\alpha)$

$a = \sqrt{r_4^2 + z_4^2}$

$a\cos(\gamma) = z_4 \rightarrow \cos(\gamma) = \frac{z_4}{a} = C$

$\hookrightarrow \sin(\gamma) = \pm\sqrt{1 - \left(\frac{z_4}{a}\right)^2} = D$

$\gamma = \text{atan2}(\pm D, C)$

$\cos(180 - \theta_3) = -\frac{a^2 - L_2^2 - L_3^2}{2 L_2 L_3} = E$

$\sin(180 - \theta_3) = \pm\sqrt{1 - (E)^2} = F$

$180 - \theta_3 = \text{atan2}(\pm F, E)$

$\boxed{\theta_3 = 180 - \text{atan2}(\pm F, E) - \text{offset}}$ 79°

$\cos(\beta) = -\frac{L_3^2 - a^2 - L_2^2}{2 L_2 a} = G$

$\sin(\beta) = \pm\sqrt{1 - G^2} = H$

$\beta = \text{atan2}(\pm H, G)$

offset

$\theta_2 = 90 - \beta - \gamma + 10.6°$

$\boxed{\theta_2 = 90 - \text{atan2}(\pm H, G) - \text{atan2}(\pm D, C)} \boxed{+ 10.6°}$

dont need?

$-\alpha = 90 - \theta_2 - \theta_3 + \theta_4$

$\boxed{\theta_4 = \theta_2 + \theta_3 - \alpha - 90}$

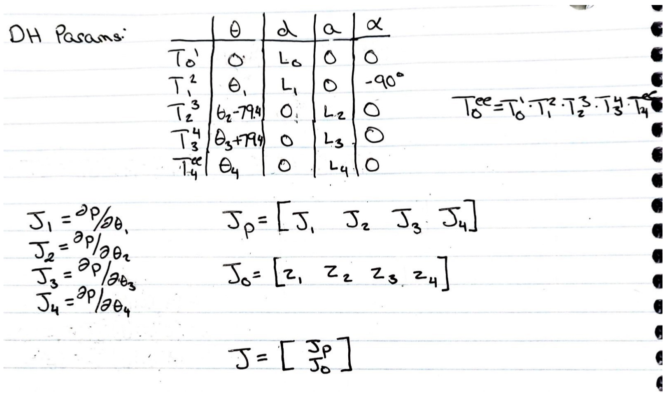
$\hookrightarrow$ should θ_4 be negative?

don't need 90°?

Figure 3: Inverse Kinematic Derivation of the Robot Arm

Fig. 3 displays the calculations made for the inverse position kinematics of the robot arm where the four joint angles are calculated based on the task space. A geometric approach was used for this derivation. Using these calculation the **ik3001()** function was developed to calculate the required joint angles from a given task space position.

B. Forward and Inverse Velocity Kinematics



DH Params:

	θ	d	a	α
T_0^1	0	L_0	0	0
T_1^2	θ_1	L_1	0	-90°
T_2^3	θ_2-79.4	0	L_2	0
T_3^4	θ_3+79.4	0	L_3	0
T_4^{ee}	θ_4	0	L_4	0

$T_0^{ee} = T_0^1 \cdot T_1^2 \cdot T_2^3 \cdot T_3^4 \cdot T_4^{ee}$

$J_1 = \partial p / \partial \theta_1$

$J_2 = \partial p / \partial \theta_2$

$J_3 = \partial p / \partial \theta_3$

$J_4 = \partial p / \partial \theta_4$

$J_p = [J_1 \;\; J_2 \;\; J_3 \;\; J_4]$

$J_o = [z_1 \;\; z_2 \;\; z_3 \;\; z_4]$

$J = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$

Figure 4: Jacobian Derivation for the Robot Arm

Fig. 4 displays the derivation for calculating the Jacobian matrix using the position vector from the transformation from the base to the end effector and partially differentiating it with respect to each of the four joints. This proved the upper half of the Jacobian. Since all the joints are revolute, for the lower half of the Jacobian, the third column of the rotation matrices for each of the joints is calculated. The rotation matrices used are from the transformation matrix from the base to the given joint frame. The derived jacobian matrix is then multiplied by the joints' velocities to calculate the end effector's velocities.

C. Camera Setup and Intrinsic Calibration

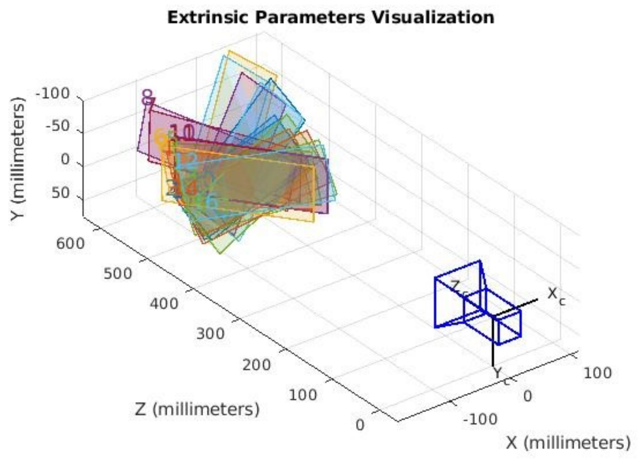


Figure 5: Extrinsic Parameter Visualization

Fig. 5 displays the extrinsic parameters and different orientations of the checkerboard for the final 20 calibration images.

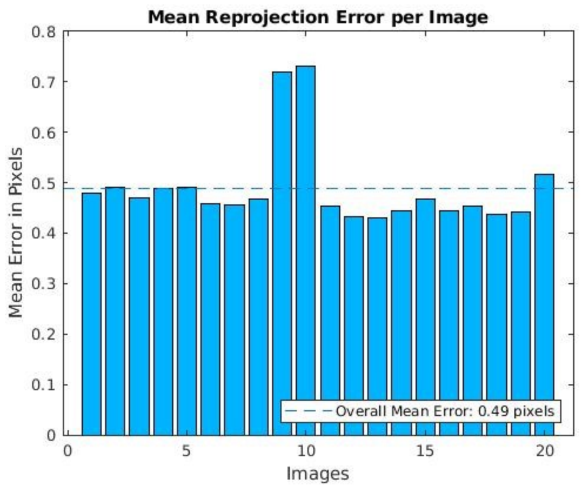


Figure 6: Histogram of Mean Reprojection Error per Image

Fig. 6 displays the mean reprojection error in pixels for each of the 20 final calibrated images of the checkerboard. The overall mean error is 0.49 pixels which is well below the target mean error of under 1 pixel.

The reproduction error displayed in the histogram is the difference between the checkerboard points detected in the calibrated image and the world points projected using the estimated camera matrix. The mean reprojection error being 0.49 pixels, which is a relatively low value, signifies the accuracy of the calibration due to how close the 3D projection is to the image coordinates. This mean was achieved by discarding approximately 7-10 images where the mean was over 1 pixel.

D. Object Detection and Classification

After experimenting with multiple image processing pipelines, the finalized pipeline included masking anything outside the checkerboard, masking the colored balls using

hsv filters, eroding, dilating and filing the masks of the ball. Anything outside the checkerboard was masked to prevent any detection of target objects outside the workspace. For masking the colored balls, the HSV color space was used for filters due to its separation of brightness from color information which is particularly useful for color-based image segmentation. Erosion of the target object masks was done to eliminate any smaller pixels that could be detected when identifying centroids. Dilation was then applied to enlarge the masks. Then, any holes in the masks were filled to improve the accuracy of the centroid.
A black ball cannot be used for the purpose of this project due to the checkerboard having black squares which would cause any color filter that is applied to detect the checkerboard squares themselves as target objects. An object's color matters as it is the feature that identifies and differentiates it from the rest of the workspace.

F. Object Localization

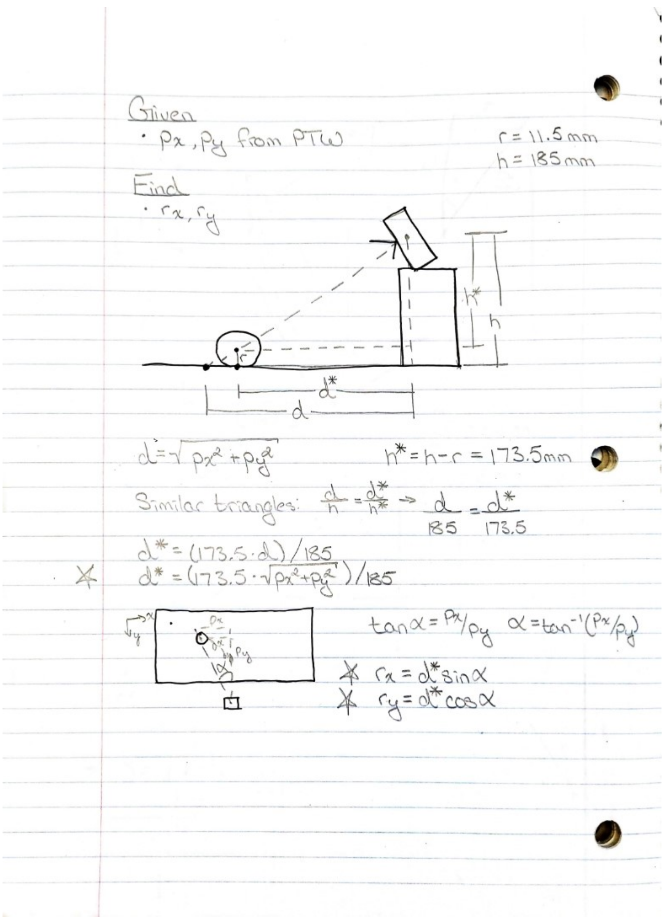


Figure 7: Calculation of Offsets for Object Localization

The object localization is extremely important in being able to accurately navigate to the ball. The error between the projected position and the real position can be close to 10mm in some locations depending on the angle. This is a large error for the arm as the gripper has a small margin to be able to successfully pick up the ball. As seen in Fig. 7 our calculations included a side view and a top view to be able to determine the offset. An additional error we found was that our transformation of the robot base to the checker base had to be extremely accurate as that also affected the position the robot would navigate to.

G. Design of Pick and Place Sorting System

For the final sorting system, a while loop was used to check whether there are any detected centroids in the image taken. This ensures that the program with autonomously run until there is no target object in the workspace. In the while loop, the image processing pipeline is followed which detects the target objects, their centroids and distinguishes them by color in order for them to be sorted in the specified locations. To ensure that any previously picked ball is not detected, a mask using the coordinates of the checkerboard was placed for anything outside of the checkerboard. After the transformation calculations, inverse kinematics is applied to calculate the joint angles requires for the robot to move just above center of the object to avoid moving it and then directly down to pick the object using the grippers. The type of trajectory chosen for the robot's motion was cubic trajectory due to the smooth and continuous motion over time. The sorting system that was selected placed each of the four colors of the balls outside each of the corners of the checkerboard.

IV. Conclusion

Throughout this lab we used various methods related to robotic manipulators to be able to complete the pick and sort task. We used forward inverse, and velocity kinematics to move between the joint and task space. All of these functions were developed and tested in previous labs. Additionally, we used trajectory planning to smoothly move between positions. Finally, we learned about computer vision and how to process and image to extract the relevant information.

All of this was integrated into a fully functional system that was successfully able to sort the ball objects based on their color. The script was constantly running and updating so that it would always use the most up to date information, and sort all of the balls.

V. Links

A. Link for the Code
https://github.com/RBE3001-A23/RBE3001_A23_Team23/releases/tag/Lab5FinalCode

B. Link for the Video
https://youtu.be/s0xQ3encRHw?feature=shared

Appendix

A. Table of Contribution

Section	Author
Introduction	Aliaa Hussein
Methodology	Aiden Deady, Aliaa Hussein
Results	Aliaa Hussein
Discussion	Aiden Deady, Aman Hiregoudar
Conclusion	Aman Hiregoudar