LAB 5: FINAL PROJECT –ROBOTIC PICK AND PLACE SYSTEM

Yichen Guo, Yuhan Wu, Haojun Feng

***Abstract* — In this lab, we will combine everything we have learned in the past seven weeks to pick and sort objects on the checkerboard by using a camera vision to return and inverse kinematics.**

I. INTRODUCTION

In Lab 1, we communicated with the robot by sending commands to the joints through a MATLAB code; in Lab 2, we used forward kinematics to calculate the robot's position in the task space given the joint variables and visualize real-time stick model in MATLAB; in Lab 3, we implemented inverse kinematics and performed trajectory generation techniques in joint space and in task space to generate a smooth trajectory from one vertex to another and in Lab 4 we implemented differential kinematics for control of the robot.

As for this lab, which is the final project, we will combine what we have learned to make a robotic pick and place sorting system. We will incorporate computer vision to identify targets, localize them with respect to the robot, move the robot towards the object, grab them, and sort them.

- Mechanical

The 3001 robot has multiple 3D printed linkages, which allows motion in three different dimensions, but does not allow the adjustment of pitch, roll, or yaw at the end effector.
Three servos with embedded encoders are used as joints for the robot arm and all three joints are referred to as the base, elbow and wrist by our team.
The base joint has a range of about π radians; the elbow had a range of $\pi/2$ radians while the wrist had a range of $3\pi/2$ radians of dimensions.
The end effector of the robot is controlled by a gripper by using a servo as well.
A checker board and a fish-eye camera are also attached to the base to help us finish the project.

- Electrical

The robot arm is power by ITSYBITSY ADAFRUIT board with 5V 16Hz control standard and a Atmega32u4 chip. It has 23 GPIO, 6 Analog-in, 1 SPI port, 1 I2C port, 1 Hardware Serial port, 4PWM port and a Micro USB port.

- Computational

We use MATLAB as our coding environment and GitHub as a relay to help to communicate with teammates as well as setting up branches to distribute works and make our codes cleaner.

- Background

The purpose of this project is to combine, expand what we had learned in the past four labs.
To accomplish this task, the robot needed to use forward and inverse kinematics to determine the location of the target and use trajectory planning to follow the path we want and finally, pick up the object.
Then the lab focused on the camera in such way that it could be used to sort objects by their colors. The camera needed to displayed as a live-feed within MATLAB and referred to the correct frame so that we could correctly determine the center of the object.
By receiving all the information from previous procedures, the robot arm is allowed to calculate the location of the object as well as the location of the end effector before reaching to that position and that is where Inverse kinematics and trajectory planning can be implemented together to create a smooth motion from the original position.
At last, combine all the procedures illustrated, we can create the pick and sort system based on the camera vision. Since the object in this task is small and light enough, we do not need to consider the weight at the end effector of the robot arm.

II. METHODOLOGY

1. Camera Setup & Intrinsic Calibration

   We first connect and set up the camera; then, we utilized a built-in app in Matlab called Camera Calibrator to perform intrinsic calibration on our camera. To calibrate, we took a series of pictures of different orientations of the checkerboard while moving camera around.

2. Camera-robot registration

   Perform a registration between the reference frame of the robot and the reference frame of the Image. And needed to be able to relate the position of objects within the field of view of the camera to robot task space coordinates. The z needs to coordinate with the origin of the checkerboard, otherwise it will be a height offset which cause the checkerboard frame to be above or below the real checkerboard plane.

3. Object Detection and Classification

   Using the image processing methods, we should determine an image processing pipeline that takes as input a frame from the camera and identifies the centroid of a solid-colored spherical tracking object and overlay a marker on the displayed video feed. We use median filter, average filter to process the picture and get solid result for the object coordinates.

4. Convert the centroid location of the balls into usable target positions for the robot to grasp.

   When we are checking coordinates with robot, we will not get the correct spot, since the camera is viewing the ball at its edge. We measured the height of the ball and the height of camera and utilize the law of similar triangles to calculate the error, which is generally around 2mm depending on the objects position.

```
GreenYtocamera = 150 - ptwGreen(2);
GreenXtocamera = 100 - ptwGreen(1);
RedYtocamera = 150 - ptwRed(2);
RedXtocamera = 100 - ptwRed(1);
YellowYtocamera = 150 - ptwYellow(2);
YellowXtocamera = 100 - ptwYellow(1);
OrangeYtocamera = 150 - ptwOrange(2);
OrangeXtocamera = 100 - ptwOrange(1);
BlueYtocamera = 150 - ptwBlue(2);
BlueXtocamera = 100 - ptwBlue(1);

%?????????????????????????????????????????????????????????????????????????????????????????????
GreenRealY = ((12/317.5)*(GreenYtocamera))+ptwGreen(2);
GreenRealX = ((12/317.5)*(GreenXtocamera))+ptwGreen(1);
RedRealY = ((12/317.5)*(RedYtocamera))+ptwRed(2);
RedRealX = ((12/317.5)*(RedXtocamera))+ptwRed(1);
YellowRealY = ((12/317.5)*(YellowYtocamera))+ptwYellow(2);
YellowRealX = ((12/317.5)*(YellowXtocamera))+ptwYellow(1);
OrangeRealY = ((12/317.5)*(OrangeYtocamera))+ptwOrange(2);
OrangeRealX = ((12/317.5)*(OrangeXtocamera))+ptwOrange(1);
BlueRealY = ((23/317.5)*(GreenYtocamera))+ptwGreen(2);
BlueRealX = ((23/317.5)*(GreenXtocamera))+ptwGreen(1);
```

Figure 1. True Coordination calculation

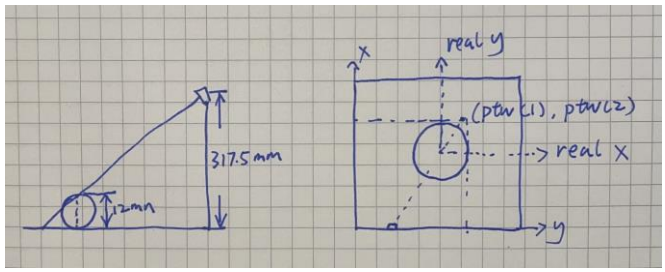


Figure 2. Illustration of our method

5. Implement the fully automated pick and place robotic sorting system by putting the coordinate input into inverse kinematics and trajectory function to complete the pick-up and sorting system.

   i. Put four balls with different color (green, red, yellow, orange) on the checkerboard

   ii. Use the camera to take a picture of the current position of the four balls.

   iii. Process the picture with different color mask and get the centroid of each ball.

   iv. Plan the Trajectory using cubic polynomial: send the robot arm above the ball, lower the arm straight down to catch the ball, rise straight up to the position above the ball, transfer the ball to the it's assigned coordinate.

   v. Repeat the step iv for each ball.

   vi. Continuously perform step ii to v, in the end of each loop, send the robot to zero position so that the camera can take the picture of the ball that has been put back.

   Overall, our method of sorting is that we will first sort the four original balls, during the process of sorting, we will put back some sorted balls back to the checkerboard, when the original sorting finished, the system will take another picture about the balls that has been put back and sort those balls again. The sorted balls with a centroid withing the range of the assigned sorted place will be excluded in the following sorting loop.

6. Extra 1: Modify camera tracking software to implement real-time object tracking.

   We wrote a loop to continuously taking pictures of the current checkerboard. Inside the loop, the code will process the picture to get the centroid of the current object using the same type of masks we used in step 3. Then we send the robot arm to the coordinate above the current centroid in each loop. Thus, if the detected coordinate of the object's centroid is changing, the arm will follow the object.

7. Extra 2: Program your robot so that it can detect, localize and grasp objects of your choice.

   In this step, our group process the picture to get the centroid of the current object, we use a larger color range in the color thresholder so it could detect random color. Then, we send the robot arm to the coordinate of the centroid and grasp the object.

III. RESULTS

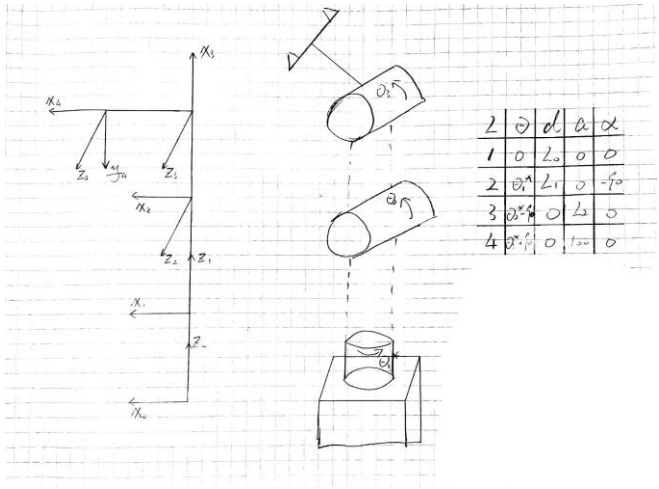


Figure 3. Forward kinematics DH table and frame

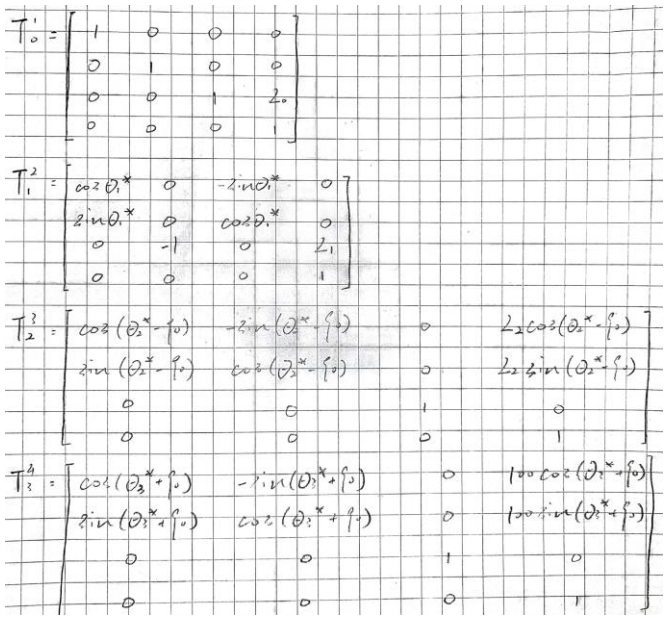


Figure 4. Forward kinematics DH table and frame

Figure 1 and 2 are the manual solution of the frames of the robot arm and the corresponding DH table, we used the solution to solve for the transfer matrix and verify the answer in the MATLAB, which are shown in Figure 3 and 4.

```
Q1_T01 = pp.dh2mat(full_dh_table_1(1,:))
Q1_T12 = pp.dh2mat(full_dh_table_1(2,:))
Q1_T23 = pp.dh2mat(full_dh_table_1(3,:))
Q1_T34 = pp.dh2mat(full_dh_table_1(4,:))

Q2_T01 = pp.dh2mat(full_dh_table_2(1,:))
Q2_T12 = pp.dh2mat(full_dh_table_2(2,:))
Q2_T23 = pp.dh2mat(full_dh_table_2(3,:))
Q2_T34 = pp.dh2mat(full_dh_table_2(4,:))

Q3_T01 = pp.dh2mat(full_dh_table_3(1,:))
Q3_T12 = pp.dh2mat(full_dh_table_3(2,:))
Q3_T23 = pp.dh2mat(full_dh_table_3(3,:))
Q3_T34 = pp.dh2mat(full_dh_table_3(4,:))

Q4_T01 = pp.dh2mat(full_dh_table_4(1,:))
Q4_T12 = pp.dh2mat(full_dh_table_4(2,:))
Q4_T23 = pp.dh2mat(full_dh_table_4(3,:))
Q4_T34 = pp.dh2mat(full_dh_table_4(4,:))
```

Figure 5. MATLAB calculation for FK 1

```
q1 = [0,0,0]';
full_dh_table_1=[0,55,0,0;
                 q1(1,1),40,0,-90;
                 q1(2,1)-90,0,100,0;
                 q1(3,1)+90,0,100,0;];

q2 = [-90.00, 86.14, 33.71]';
full_dh_table_2=[0,55,0,0;
                 q2(1,1),40,0,-90;
                 q2(2,1)-90,0,100,0;
                 q2(3,1)+90,0,100,0;];
q3 = [0,90,-90]';
full_dh_table_3=[0,55,0,0;
                 q3(1,1),40,0,-90;
                 q3(2,1)-90,0,100,0;
                 q3(3,1)+90,0,100,0;];
q4 = [90,0,0]';
full_dh_table_4=[0,55,0,0;
                 q4(1,1),40,0,-90;
                 q4(2,1)-90,0,100,0;
                 q4(3,1)+90,0,100,0;];
```

Figure 6. MATLAB calculation for FK 2

We plugged in various numbers to prove our answers in MATLAB and it turned out to be consistent.

Similar for solving for inverse kinematics, we still had a manual solution based on geometric approach, because in our frame drawing, since all the joints are revolute joints, it is easier to find out those variables on the graph. Manual solution is presented on Figure 5 and Figure 6 and 7 are MATLAB calculation for inverse kinematics.

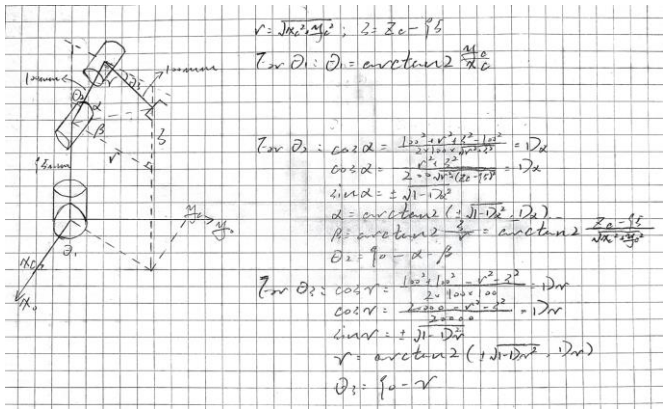


Figure 7. Manual solution for inverse kinematics

```
        % This method takes a 3x1 task space position vector as the input
        % and returns a set RBE 3001 A-Term 2021 -Lab 33of corresponding
        % joint angles (i.e. q1, q2, q3) that would make the robot's
        % end-effector move to that target position with elbow up path. The
        % position vectors are in mm, joint angles are in degrees.
        function q_v = ik3001(self,p_v)
            x = p_v(1);
            y = p_v(2);
            z = p_v(3);
            %through the error if the task space is unreachable.
            if (p_v(1)>=200 || p_v(2)>=200 || p_v(3)>=295)
                error('end effector position is out of workspace')
            end
             % Calculated formular.
            %D_q1 = x/sqrt(x^2+y^2);
            D_alpha = (x^2+y^2+(z-95)^2)/(200*sqrt(x^2+y^2+(z-95)^2));
            D_q3 =(20000-x^2-y^2-(z-95)^2)/20000;
            % calculate each joint angle using atan2.
            q1 = atan2d(p_v(2),p_v(1));
            %q1 = atan2d(sqrt(1-(D_q1)^2), D_q1);
            alpha = atan2d(sqrt(1-(D_alpha)^2), D_alpha);
            beta = atan2d((z-95),sqrt(x^2 + y^2));
            q2 = 90 - (alpha + beta);
            q3 = 90 - atan2d(sqrt(1-(D_q3^2)), D_q3);
            %In case we need it, here is the elbow down path.
%             q1 = atan2(-sqrt(1-D_q1^2), D_q1);
%             q2 = atan2(-sqrt(1-D_q2^2), D_q2);
%             q3 = atan2(-sqrt(1-D_q3^2), D_q3);
            q_v = [q1; q2; q3];
        end
    end
end
```

Figure 8. MATLAB calculation for IK 1

```
myHIDSimplePacketComs=HIDfactory.get();
myHIDSimplePacketComs.setPid(pid);
myHIDSimplePacketComs.setVid(vid);
myHIDSimplePacketComs.connect();

% Create a PacketProcessor object to send data to the nucleo firmware
pp = Robot(myHIDSimplePacketComs);
m = Model(pp);
traj5 = Traj_Planner(pp);
% This is for sign-off 1

%return a set of position for[100;0;195]in 1x3 vector;
v0 = pp.ik3001([100;0;195])
%return a set of position for [-2;2;273] in 1x3 vector;
v1 = pp.ik3001([-2;2;273])
%return a set of position in rad for [-74;-18;265] in 1x3 vector;
v2 = pp.ik3001([-74;-18;265])

fk0 = pp.fk3001(v0');
fk1 = pp.fk3001(v1');
fk2 = pp.fk3001(v2');

p0 = [fk0(1,4),fk0(2,4),fk0(3,4)]
p1 = [fk1(1,4),fk1(2,4),fk1(3,4)]
p2 = [fk2(1,4),fk2(2,4),fk2(3,4)]
```

Figure 9. MTLAB calculation for IK 2

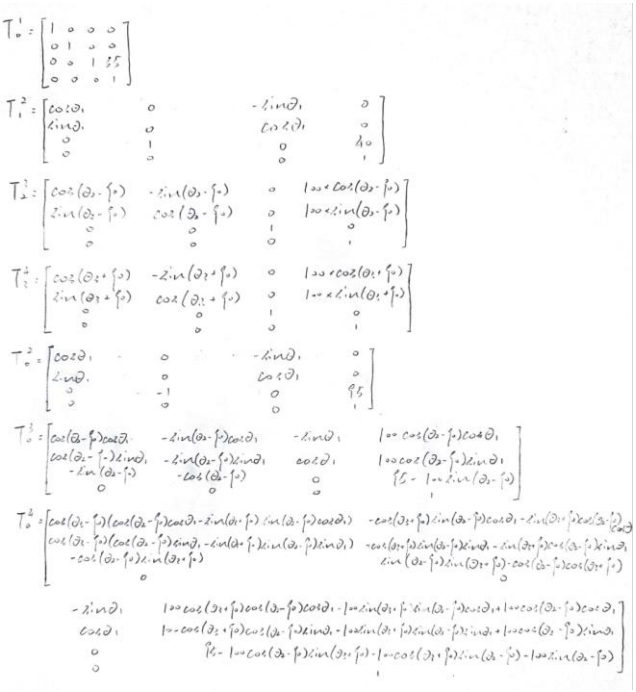


Figure 10. Manual solution for forward velocity kinematics

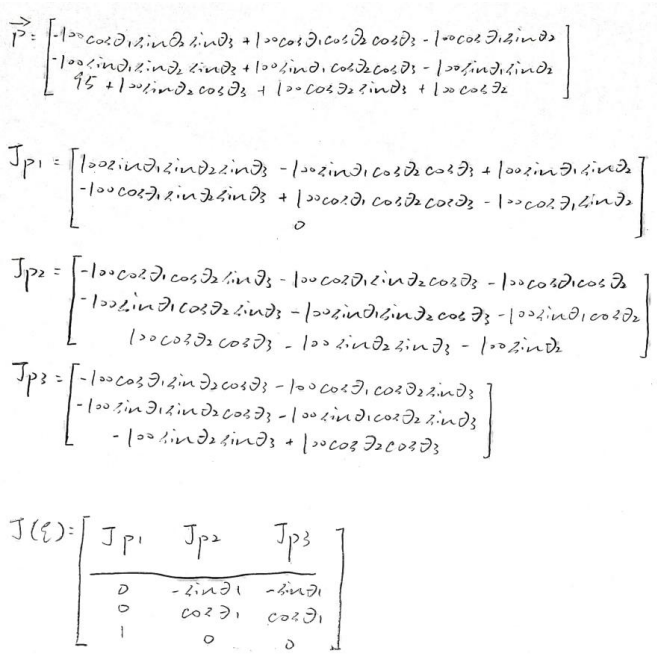
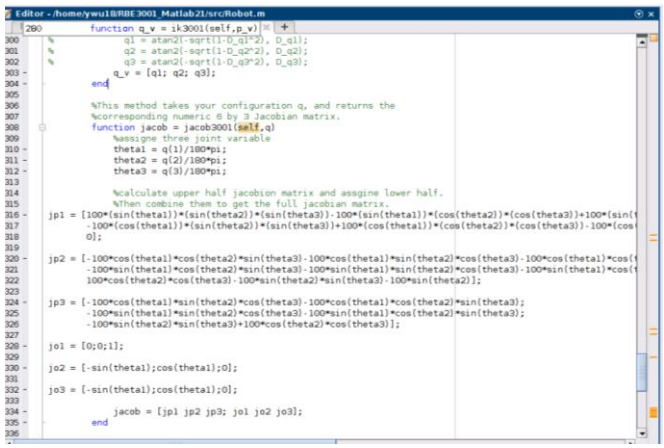


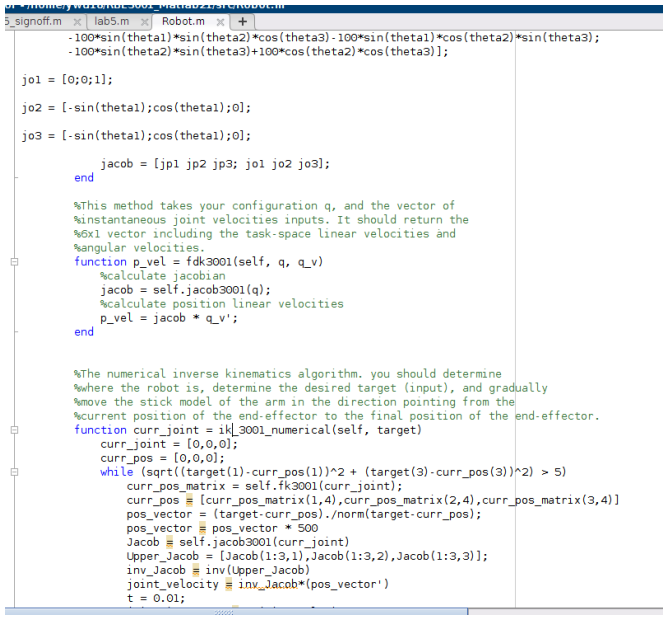
Figure 11. Manual solution for Jacobian

Similar to previous procedures, we used transfer matrix and end point vector to calculate forward velocity kinematics and derivative the end vector with respect to each joint to calculate upper Jacobian. We then validate our solution with MATLAB shown in Figure 10.



```
Editor - /home/ywu18/RBE3001_Matlab21/src/Robot.m
280         function q_v = ik3001(self,p_v)
300     %           q1 = atan2(-sqrt(1-D_q1^2), D_q1);
301     %           q2 = atan2(-sqrt(1-D_q2^2), D_q2);
302     %           q3 = atan2(-sqrt(1-D_q3^2), D_q3);
303 -           q_v = [q1; q2; q3];
304 -       end
305
306         %This method takes your configuration q, and returns the
307         %corresponding numeric 6 by 3 Jacobian matrix.
308         function jacob = jacob3001(self,q)
309             %assigne three joint variable
310 -           theta1 = q(1)/180*pi;
311 -           theta2 = q(2)/180*pi;
312 -           theta3 = q(3)/180*pi;
313
314             %calculate upper half jacobion matrix and assgine lower half.
315             %Then combine them to get the full jacobian matrix.
316 -   jp1 = [100*(sin(theta1))*(sin(theta2))*(sin(theta3))-100*(sin(theta1))*(cos(theta2))*(cos(theta3))+100*(sin(t
317 -       -100*(cos(theta1))*(sin(theta2))*(sin(theta3))+100*(cos(theta1))*(cos(theta2))*(cos(theta3))-100*(cos(
318 -       0];
319
320 -   jp2 = [-100*cos(theta1)*cos(theta2)*sin(theta3)-100*cos(theta1)*sin(theta2)*cos(theta3)-100*cos(theta1)*cos(t
321 -       -100*sin(theta1)*cos(theta2)*sin(theta3)-100*sin(theta1)*sin(theta2)*cos(theta3)-100*sin(theta1)*cos(t
322 -       100*cos(theta2)*cos(theta3)-100*sin(theta2)*sin(theta3)-100*sin(theta2)];
323
324 -   jp3 = [-100*cos(theta1)*sin(theta2)*cos(theta3)-100*cos(theta1)*cos(theta2)*sin(theta3);
325 -       -100*sin(theta1)*sin(theta2)*cos(theta3)-100*sin(theta1)*cos(theta2)*sin(theta3);
326 -       -100*sin(theta2)*sin(theta3)+100*cos(theta2)*cos(theta3)];
327
328 -   jo1 = [0;0;1];
329
330 -   jo2 = [-sin(theta1);cos(theta1);0];
331
332 -   jo3 = [-sin(theta1);cos(theta1);0];
333
334 -           jacob = [jp1 jp2 jp3; jo1 jo2 jo3];
335 -       end
336
```

Figure 12. MATLAB calculation for Forward velocity kinematics



```
5_signoff.m | lab5.m | Robot.m
        -100*sin(theta1)*sin(theta2)*cos(theta3)-100*sin(theta1)*cos(theta2)*sin(theta3);
        -100*sin(theta2)*sin(theta3)+100*cos(theta2)*cos(theta3)];

jo1 = [0;0;1];

jo2 = [-sin(theta1);cos(theta1);0];

jo3 = [-sin(theta1);cos(theta1);0];

            jacob = [jp1 jp2 jp3; jo1 jo2 jo3];
        end

        %This method takes your configuration q, and the vector of
        %instantaneous joint velocities inputs. It should return the
        %6x1 vector including the task-space linear velocities and
        %angular velocities.
        function p_vel = fdk3001(self, q, q_v)
            %calculate jacobian
            jacob = self.jacob3001(q);
            %calculate position linear velocities
            p_vel = jacob * q_v';
        end


        %The numerical inverse kinematics algorithm. you should determine
        %where the robot is, determine the desired target (input), and gradually
        %move the stick model of the arm in the direction pointing from the
        %current position of the end-effector to the final position of the end-effector.
        function curr_joint = ik3001_numerical(self, target)
            curr_joint = [0,0,0];
            curr_pos = [0,0,0];
            while (sqrt((target(1)-curr_pos(1))^2 + (target(3)-curr_pos(3))^2) > 5)
                curr_pos_matrix = self.fk3001(curr_joint);
                curr_pos = [curr_pos_matrix(1,4),curr_pos_matrix(2,4),curr_pos_matrix(3,4)]
                pos_vector = (target-curr_pos)./norm(target-curr_pos);
                pos_vector = pos_vector * 500
                Jacob = self.jacob3001(curr_joint)
                Upper_Jacob = [Jacob(1:3,1),Jacob(1:3,2),Jacob(1:3,3)];
                inv_Jacob = inv(Upper_Jacob)
                joint_velocity = inv_Jacob*(pos_vector')
                t = 0.01;
```

Figure 13. MATLAB calculation for Inverse velocity kinematics

Figure 11 shows the steps we made to find Inverse velocity kinematics in MATLAB. Since it is hard for calculating it manually, we only used the MATLAB approach to get the answer.

- Final project process

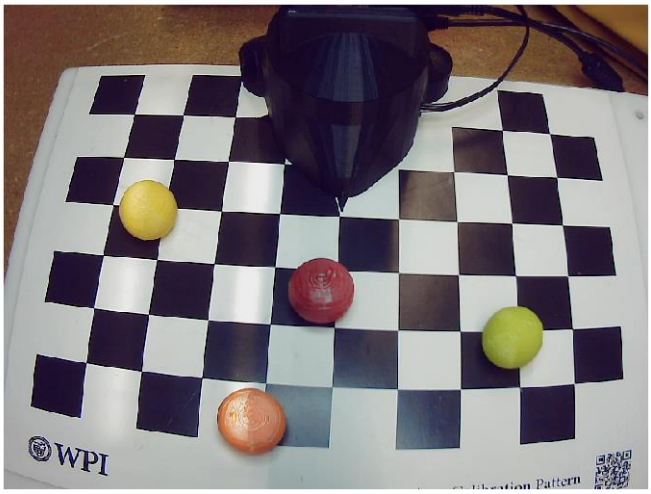


Figure 14. Camera image

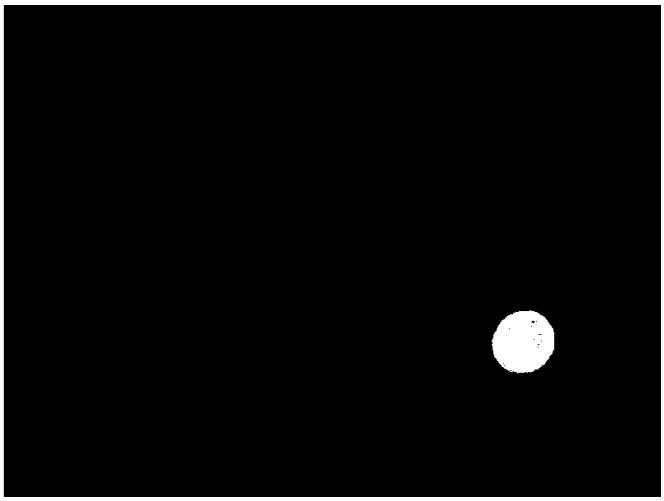


Figure 15. greenBW after using greenMask

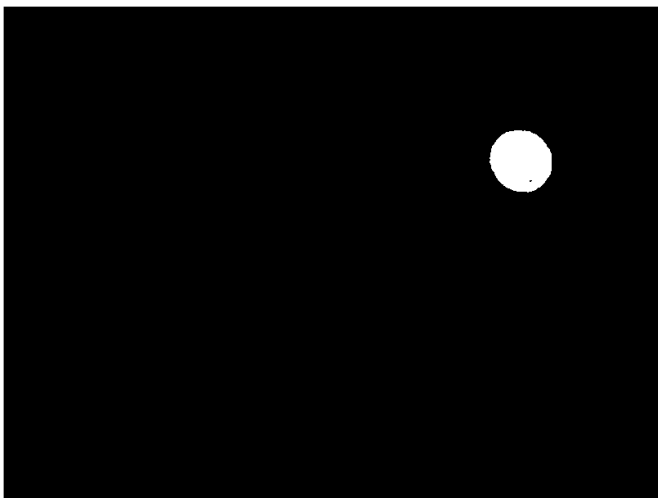


Figure 16. greenMed after using medfilt2

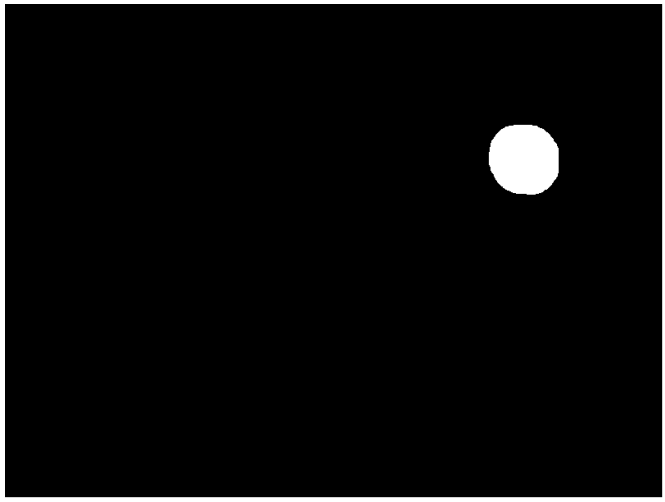


Figure 17. greenAvg after using average filter

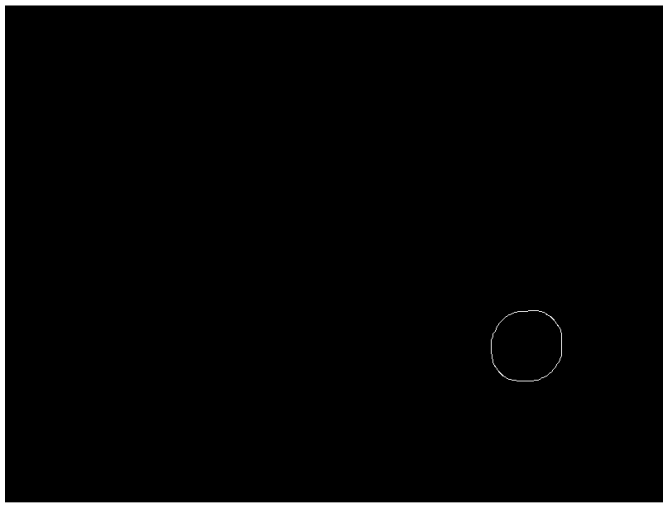


Figure 18. greenEdge after using edge

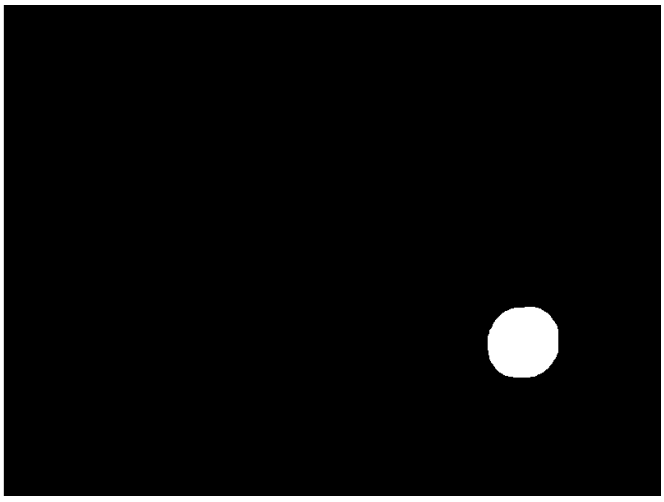


Figure 19. greenFill after using infill

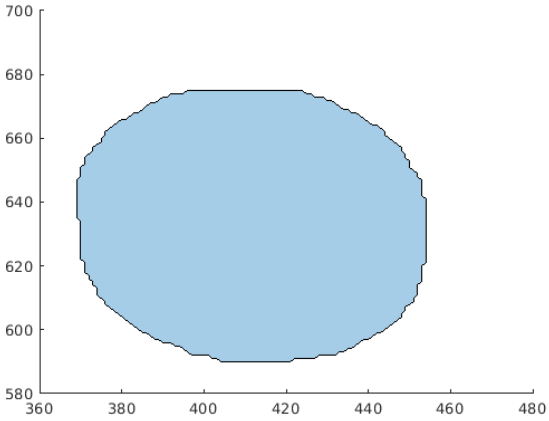


Figure 20. greenPoly (coordinate in camera frame)

```
greenCenterY =

  411.7621

greenCenterX =

  632.8634
```

Figure 21. Green ball centroid

IV. Discussion

During the process of Camera calibration, we have a histogram of reprojection error. As it was said in our lab documents: 'The reprojection error is the distance between the checkerboard points detected in a calibration image, and the corresponding world points projected using the estimated camera matrix.' We excluded the photo with a reprojection error larger than 0.8 and get our overall mean error at: 0.42.

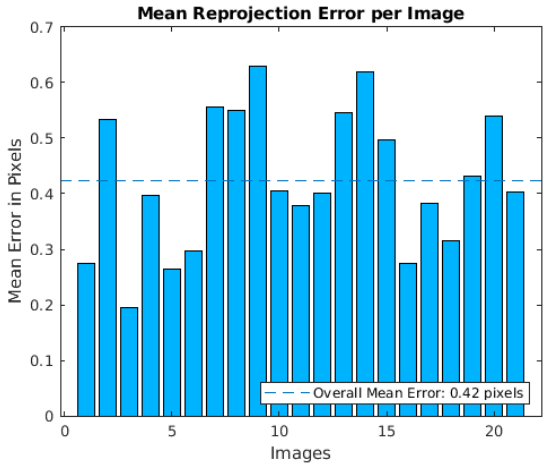


Figure 22. Reprojection error histogram

During our calibration, I think the camera will calculate the pixel coordinates and locate each checkerboard square and record their location. With we are setting the size of the square to be 25mm, the camera will distort it's view to fit the pixel coordinates with the size of the square we selected and thus calibration s

As seen in Figure 14 to 20, we utilized median filter and average filter to process the photo, and get a polygon with nearly round shape, which indicate that our system is accurate.

We selected HSV color space when applying mask since the four colors are more distinctive in HSV color space rather than in RGB color space. Since we use color to detect and separate object, it will be nearly impossible to detect a black ball from a black and white checker frame.

For the last step, sorting approach and code logic is described in Methodology chapter. We selected linear trajectory since the arm will work in straight line and will not affect another object.

V. Conclusion

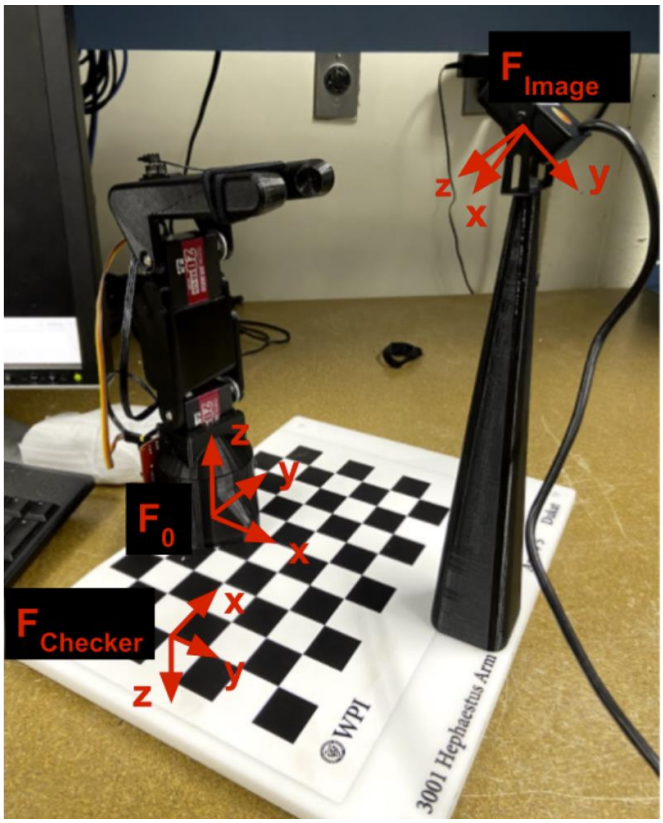
In this final project, we successfully combined everything we learned through the term, starting from frame analysis, forward / inverse kinematics, trajectory plan, Jacobian to Camera vision. We successfully implemented a pick and sort system based on these theories. We did not re-calculate kinematics-relate systems, because they had already been validated through the term while camera vision was the most concerning thing in this project and that was where we spent

the most time on.
Similar to other sensing systems, environment condition is the most important determinant for calibration. We picked the most proper camera vision calculation method to detect color in complicated environment and managed to use the algorithm to pick up random objects with respect to colors.
Furthermore, we also managed to accomplished the real-time picking system. The robot arm can follow the trajectory of an object, but due to the light conditions and other interferences, sometimes some glitches occur, but after re-calibrating color set ups of the camera, it was contained in an acceptable range and unlikely to happen unless extreme condition occurs.

VI. Contribution

	Planning	Coding	Experimenting	Analyze results	Writing	Video
Yichen Guo	33%	33%	33%	33%	33%	33%
Yuhan Wu	33%	33%	33%	33%	33%	33%
Haojun Feng	33%	33%	33%	33%	33%	33%

APPENDIX A: FRAME CLARIFY



APPENDIX B: AUTHORSHIP

Introduction	Yichen Guo / Yuhan Wu
Methodology	Yichen Guo / Haojun Feng
Results	Haojun Feng
Discussion	Yuhan Wu / Haojun Feng
Conclusion	Haojun Feng / Yichen Guo

APPENDIX C: CODE LINK & VIDEO LINK

https://github.com/RBE300X-Lab/RBE3001_Matlab21/releases/tag/Final_version_of_the_final_project_code_submission

https://youtu.be/2TNCAHdsym4