Development of a Vision Controlled 3DOF Robotic Arm Pick and Place System

Casey Gosselin, Amber Lindberg, Aaron Longo

RBE 3001 UNIFIED ROBOTICS III: ACTUATION C21 Team 9

3/18/2021

***Abstract* — This project details the creation of a computer vision controlled robotic arm system. An even driven state machine was coded using MATLAB on a Linux workstation. Test trials resulted in the robot being able to detect and localize balls before sorting them by color. The system was then able to perform dynamic object tracking and unique object manipulation.**

Terms – Computer Vision, Kinematics, MATLAB, Ubuntu

INTRODUCTION

Built from the ground up, the object of this challenge was to program a three degree of freedom arm (3DOF) to use camera input to detect the location of brightly colored balls, pick the balls up without interfering with other balls and remove them from the field, placing them in their specified target location (Figure 1).

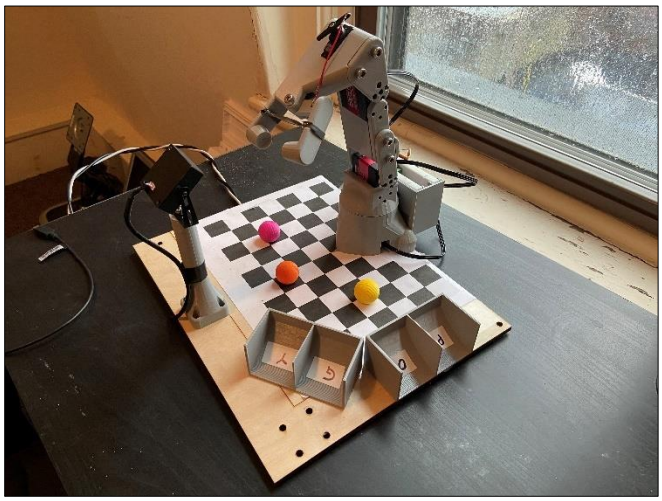

FIGURE 1: THE FINAL ROBOT FIELD SET UP.

This project implements mechanical skills through the construction of the robot arms and theory of joint-space control, electrical skills using the Hephaestus Brain control board, and programming skills using MATLAB as an IDLE. The Hephaestus Arm is a 3 DOF arm consisting of two links attached to a rotating base. The third link has a servo-powered gripper for manipulating objects on the field, which is used to pick up the 3D printed balls. This gripper servo is connected to the interface board using PWM. The arm uses serial communication among three smart servos to actuate the links which is powered by 7V connected to the line driver. Controlling the arm is the Hephaestus Brain which contains the line driver and the Itsy Bitsy M4. This board serves as the interface board for physical arm control and calibration. The firmware stored on this processor communicated with the Linux Workstation using MATLAB scripts through USB to receive signals. Along with this, a camera was connected to the Linux PC using USB connection to send information that could be used to the robot. A full diagram of the system architecture can be found in Appendix G.

There were two different types of space used within this project. One type of space was joint space. Joint space sent joint values to the robot, where the robot used forward kinematics to move. The second type of space used was task space which used inverse kinematics for the robot to move. For inverse kinematics, the robot was told which coordinates to move to, and solved for the joint angles that would get it to those positions.

To accomplish the mission of sorting the balls, the camera used image processing to detect where each ball was in task space in relation to the field, and then relayed that information to the arm. From there, the arm used inverse kinematics to determine how to reach the correct location of the ball. The robot needed to solve for the theta values of the

joints that are needed to convert the robot's tip position from joint space to task space.

METHODOLOGY

The first step for the project was to set up the webcam for computer vision tracking. Utilizing the built in MATLAB camera calibration app, many pictures of the checkerboard had to be taken. These pictures would then be calibrated using a standard or fisheye calibration function to determine the intrinsic parameters for each camera. When calibrating the robot, it was always calibrated in its "home" position, as shown in Figure 2. This way the robot knew where it was starting every single time it was powered up and would allow the code to work on multiple arms despite small variations. This ensured that the robot started in the correct spot and the robot tip was in the same place in real life that the computer thought it was.

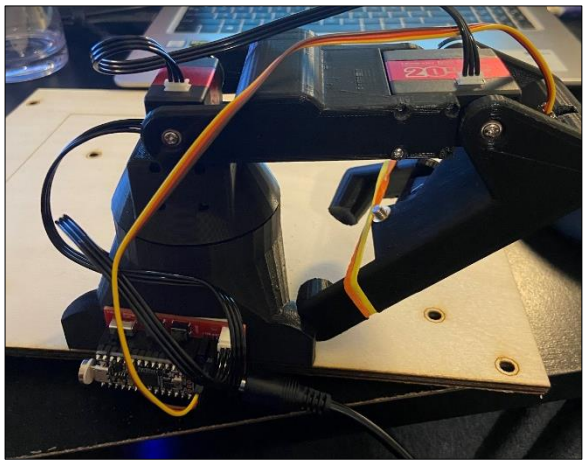

FIGURE 2: ROBOT IN "HOME" CONFIGURATION

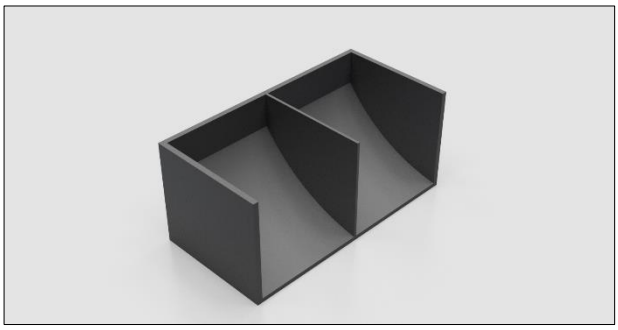

FIGURE 3: BINS USED TO DROP OFF THE BALLS

To drop off the balls in the correct location and ensure they rolled off the field and out of the view of the camera, the team designed bins that were 3D printed (Figure 3). Due to the reachable workspace of the robot, the bins were separated into two, two bin containers. The bins had different sections that corresponded for each color ball. The container contained a curved ramp that ensured the ball would move outside of the camera's views so that the robot did not try and pick up the ball again.

One of the first things done when receiving the project was creating an event driven state machine which helped simplify the code writing process. An event driven state machine helps prevent errors as the one event triggers another. This prevented the robot from skipping to the next step before it completed the previous one.

The next task completed was camera calibration. Within the Computer Vision Toolbox, there is an app called camera calibration that the team used to calibrate their cameras. The camera was moved around the board held so that all of the field squares could be seen, taking pictures every five seconds. About 35 pictures were taken per team member. Once all the pictures were taken, the team entered the size of each square in millimeters into the program. An error graph would pop up and the highest error reading pictures were deleted until there were only twenty pictures left. After completing this process, the camera was marked as a fisheye, and exported to function.

Once the camera was fully calibrated, the next task that the team tackled was creating an image processing pipeline. The way that the image processing pipeline works is by taking a snapshot using the camera, the image is undistorted, and a mask is put over it so only the checkerboard workspace is observed. Then an HSV filer is applied to the masked image and centroid values are derived from the image after a bit of filtering to get rid of any unwanted noise.

Using the information sent from the camera, several new functions where created. The first new function created was ik_3001_final num(). This function was very similar to a previous function that was created which used inverse velocity kinematics to control the movement of the arm. Inverse velocity kinematics uses inverse kinematics and the current position of the arm, along with a speed coefficient to determine what path to follow and how fast the robot will travel. Inverse kinematics calculations can be found in Appendix B. Other functions that were created in order to simplify the robot's code and make it easier to read were moveAboveBall(), moveToPickup(), moveGripper(), raiseArm(), and moveToDropPos().

Once these functions were created, a StateMachine class was created in MATLAB. This is where the robot changed from state to state. Several states utilized the use of flags, to keep track of the robot's path. In the main code file, a StateMachine was instantiated, which allowed the robot to move through its desired path and sort the balls by their perceived color.

Once the robot was able to sort the balls by color, continuous scanning was implemented into the design. The continuous scanning allowed for the ball to move on the robot, and the robot to sense that the ball was no longer in the position it thought it was and adjust accordingly. When the robot was able to use continuous scanning reliably, the team then tackled the challenge of picking up and object that was a different shape than the balls. The team used image processing to locate a mozzarella stick on the field and dunk it into marinara sauce. Once the stick was dunked, it was removed from the sauce and the arm was outstretched so that it was off the field.

RESULTS

During initial testing, it was found that the standard camera mount was too small. This caused the front edge of it to obstruct the camera view. Due to the camera's field of view, the camera needed to be raised about 2 cm higher than the stock camera stand reach. To fix these issues, a new camera stand was designed using CAD, as shown in Figure 4. The stand is taller and thinner at the top to capture the whole field while not obstructing the camera view.

FIGURE 4 : NEW CAMERA STAND DESIGNED AND

The final event-driven state machine has 5 classes, those being Robot, StateMachine, Camera, State, and Event Timer. It transitions between 4 defined states: IDLE, SCAN, MOVE, and GRASP.

When calibrating the cameras, a fisheye correction was used. This provided the most accurate results when using the points2world function. Along with the calibration tool, the lenses of each camera had to be adjusted until most of the checkerboard was in focus. The team found that an average of 20 images provided the most accurate results, with an average pixel error of about .5 pixels.

The vision processing occurs within the Camera class. Initially, HSV masks are run for each ball color. HSV masks were chosen because it was more reliable in different lighting conditions. Then, a noise reduction filter is run to remove any noise or stray pixels, as shown in Figure 5. Finally, region props are run on each ball to detect the centroid of a ball in pixel coordinates. These pixel coordinates are transformed into world coordinates using built in MATLAB functions.

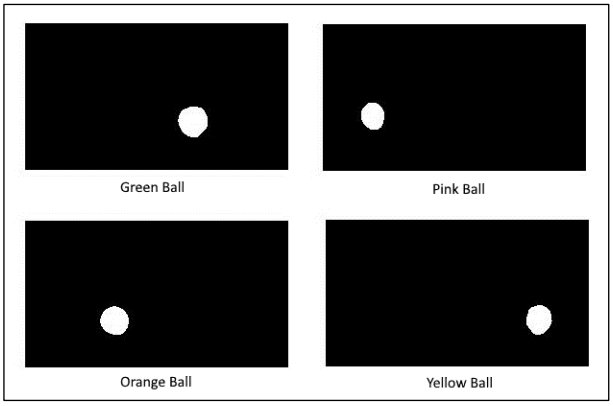

FIGURE 5: THE OUTPUT FROM THE MEDIAN FILTER FOR EACH OF THE BALLS

When picking up balls, geometric adjustments had to be made to account for picking up each ball. Essentially, the camera sees the centroid of each ball, but projects the location of it behind the ball, because the height of the centroid is equal to the radius of each ball. To calculate the true position of each ball, a function was created that accepted positional coordinates in relation to the base of the robot. From these, the coordinates were transformed to be in relation to the camera, where:

$$x_{camera} = (y_{robot} + 100) - 99$$

$$\mathrm{y_{camera}} = 151 - (x_{robot} - 50)$$

After determining the location of the ball relative to the camera, the angle from the projection to the camera lens was calculated.

$$\mathrm{p} = \sqrt{x_{cam}^2 + y_{cam}^2}$$

$$\theta_{cam} = \tan^{-1}\frac{h_{camera}}{p}$$

Once this angle was determined, the concept of similar triangles was used to determine where the height of the position vector is equal to the radius of a ball. Once this horizontal distance *a* along *p* was calculated, the coordinates

could be transformed back into coordinates relative to the robot. To calculate the new x and y coordinates of the robot, a base angle β had to be calculated.

$$a = \frac{r_{ball}}{\tan(\theta_{cam})}$$

$$\beta = \tan^{-1}\frac{x_{cam}}{y_{cam}}$$

$$x_{botNew} = (151 - (p - q)\cos(\beta)) + 50$$

$$y_{botnew} = ((p - a)\sin(\beta) + 99) - 100$$

However, the gripper on the robot does not open symmetrically, meaning the robot must be shifted to one direction to allow for maximum clearance between the fixed side of it. The following equations were used to convert the balls task space position into polar coordinates, so the angle of joint 1 on the robot could be adjusted.

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

$$\theta_{\mathrm{polar}} = \tan^{-1}\frac{y_{botnew}}{x_{botnew}}$$

Once these were determined, theta was adjusted by 5 degrees to offset the gripper.

$$\mathrm{x_{final}} = r * \cos(\theta_{polar} + 5)$$

$$\mathrm{y_{final}} = r * \sin(\theta_{polar} + 5)$$

As a result, the true and optimal pickup position of each ball could be sent to the robot. Without these calculations, the tip of the robot would move to a task space position up to a centimeter off, where the error was the worst as the edged of the checkerboard. For a diagram of the calculations, see Appendix E. For ball deposition, custom boxes were made for the balls, as seen in Figure 3. To assist in removing them the field or camera's view, they had ramps on the backside to roll the balls away.

To pick up and manipulate objects, functions had to be written for the gripper servo. This function was written using the existing write function for the servo. Since the gripper servo communicates via PWM comparted to UART like the large servos, float datatype communication was substituted for bytes.

After combining everything together, the robot was successfully able to manipulate balls of every color with reliable accuracy and consistency. For robot motion, the inverse velocity kinematics algorithm was utilized. The previously created function was modified to take in a final position in task space compared to using ginput to determine the final position. The ball boxes were attached to the field using hot glue and were assigned (left to right from the robot's perspective) pink, orange, green, and yellow.

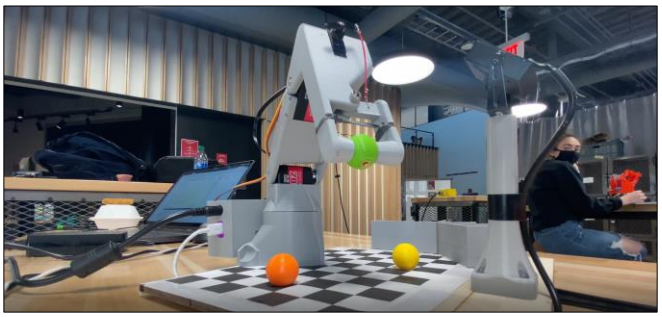

FIGURE 6: THE ROBOT MANIPULATING A BALL.

Due to the way the state machine works, the robot was also able to perform dynamic object tracking for the balls. Since every iteration of the state machine takes a new picture for ball detection, the robot could locate a ball if it had been moved. The constant scanning was implemented by switching to the SCAN state and scanning for the robot as it moved through its path. If the robot realized that the ball had moved, it would stop and change its direction towards the new location of the ball. Since dynamic tracking did not involve picking up the ball, portions of the state machine involving object manipulation were removed.

For extra credit, the team modified the existing state machine to allow for the manipulation of unique objects. The unique object was chosen to be a mozzarella stick, which the arm would move to a container of sauce in the corner of the field. The centroid calculations for the mozzarella stick had to change due to the difference in size compared to the balls. As seen in Figure X, the robot had to pick up each stick from the edge so it could rotate to a vertical position. A new mask specifically for a mozzarella stick was also made.

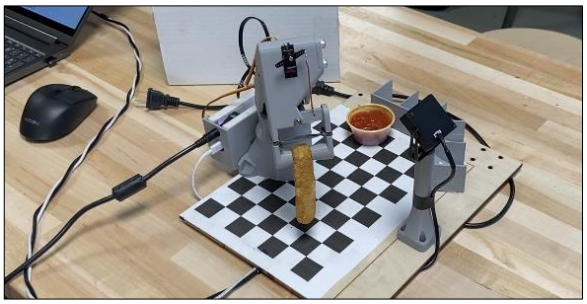

FIGURE 7: THE ROBOT GRASPING THE MOZZARELLA STICK

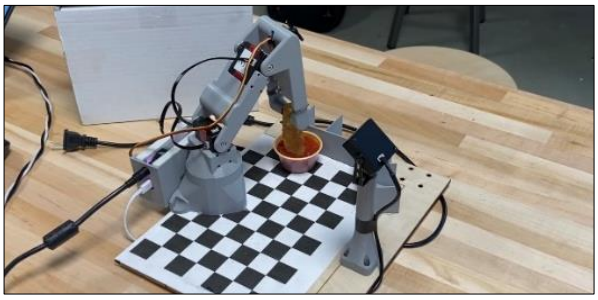

FIGURE 8: THE ROBOT DUNKING THE MOZZARELLA STICK IN MARINARA

DISCUSSION

For this project, the event-driven state machine worked well. The IDLE state referred to the period when the robot was starting up or shutting down. The SCAN state was when the robot was searching the field for objected to pick up. The MOVE state dealt with all movement of the three main smart servos, whether it was moving to a ball or to a drop off position. Finally, the GRASP state was the state where the robot would move the gripper, either to open it or close it.

Starting in the IDLE state, the robot would wait for an internal timer to expire. The timer gave the robot enough time to calibrate the camera and start up the robot. Once the timer expired, the state would change to SCAN. Within the SCAN state, the robot would undergo the image processing pipeline to find the balls. Once a ball was detected, it would travel to the MOVE state where it would begin a series of movements to get in position of the ball.

At the beginning of the MOVE state, if the robot were not above the ball and robot did not have possession of a ball, it would move to above the first ball. Once the robot was in place above the ball, it would slowly move down, surrounding the ball with each side of the gripper. Once the ball was between the gripper, the robot switched to the GRASP state, where the servo closed around the ball. Once the ball was in the robot's possession, the robot would switch back to the MOVE state. It would then raise the arm directly up. This movement made it so that the arm did not knock off other balls when moving towards the drop off location. The drop off location was determined by the color of the ball that the robot was in possession of. Once the ball had been raised a set amount, it would travel to its specified drop off position. Once it had reached its drop off position, it would switch back to the GRASP state and open its gripper. The ball would fall into the 3D printed ramp. From there, it would move back into the SCAN state, searching for another ball. If it ran out of balls, the robot shutdown, and switched back into the IDLE state.

When creating the state machine, the team realized that four states was enough states to accomplish the desired sorting method. The team utilized creating functions and flags to ensure that the robot was travelling through each section of the state machine correctly. To do this in MATLAB, a state machine class was created. The state machine has a robot object and a camera object. The camera object had functions which output the actual location of the ball. This information was fed to different functions within the robot object that were created to travel to the desired location based off the balls position.

Timing wise, the state machine depended on a series of pauses to run properly. This was determined to be due to the speed at which MATLAB runs scripts vs. how quickly the robot could respond to commands.

When calibrating the camera, the position of the checkerboard was extremely important. Since the MATLAB app recognized the black squares, the black ball could not be used. When searching for black squares, MATLAB creates a matrix of color values corresponding to the checkerboard. Since it could be assumed that the checkerboards are perpendicular, the matrix can be manipulated until the color values represent a straight line. The changes to the initial matrix represented the intrinsic parameters for each camera. Since these cameras have a wide field of view, the fisheye calibration option proved to yield the best conversion results, with an error or approximately ± 1mm.

The image processing pipeline proved to be very sensitive to the lighting conditions that the robot was in. This was because the HSV masks block out colors, allowing specific ones in. If the brightness or color of the light changed, the masks wouldn't work as intended. After initial tests, a field mask also had to be drawn. This blocked any exterior objects from being detected by the camera. Since only one object of each color could be on the field at a time, blob detection didn't have to be run to assign each object an ID.

When detecting balls, it was thought that the camera could also assign the balls to a color name, such as 'red'. Due to the way MATLAB matrices work, each color had to be

assigned to a number instead, where 1 = green, 2 = orange, 3 = pink, and 4 = yellow.

The ik_3001_final_num() function, used the information about the target ball's location in order to determine its movement. This function was the base function that was used inside of all the other new robot's movement functions.

To create trajectories for the robot motion, it was decided that the inverse velocity kinematics algorithm would be used. The team determined that this movement created the smoothest path of motion when compared to the cubic trajectory or quintic polynomial trajectory method. This movement also allowed the team to control the speed at which the robot moved using a speed coefficient, which was very helpful when slowly approaching the location of the balls.

When using the inverse velocity kinematics algorithm, small errors in joint motion were noticed in joint 3 of the robots. If the robot were moving a ball towards a drop box, joint 3 would sometimes make arcing motion before reaching the setpoint. This could be due to friction within the motor, the motor PID values, or imperfections in the robot's construction. Normally, this trajectory didn't cause any issues unless the tip came close to hitting the camera or a ball. This was solved by adjusting the task space setpoint.

The moveAboveBall() function used the ik_3001_final_num() function within it to move the ball slightly above the location of the ball. The next function, moveToPickup(), moves the arm from above the ball, slowly down to the position of the ball. The next function created was moveGripper(), which moved the gripper servo based on the value entered into the function. The value of the servo ranged from 0-180 degrees. A low number would close the gripper, and a high value would open the gripper. This function was created very similarly to how the smart servos were coded, only instead of using writeFloats() in order to communicate to the device, it used writeBytes(). The next function, raiseArm(), raised the arm of the robot in only the Z-direction. This approach was taken so that when the robot was bringing the ball to the drop position, it would not collide with any other balls on the field. Although this was made to prevent collision, the way that the state machine works, the camera re-scans the field every time that the arm deposits a ball. If a ball were to get accidently knocked, it would not be affected as its old location gets overwritten with every scan. The final new function that was created in the robot class was moveToDropPos(). This function moved the ball to its designated drop position based on the color of the ball. During the image processing pipeline, the balls are written to an array which includes an ID, based on the color of the ball. Each ball color was assigned a drop position which allowed for the ball to be moved to the correct location based off the color of the ball.

Due to the structure of the state machine, dynamic object tracking was easily achievable. However, the speed of it was slow compared to normal robot motion. Due to the time MATLAB took to process each image taken by the camera, the robot would only move with several second intervals. The team also realized when testing the movement that the color of their hand would often accidently get tracked by the camera as a ball. To counteract this, the team pushed the ball around with a screwdriver.

When trying to determine what differently shaped object the robot should pick up, the team had to make sure that the object could fit between the claw of the gripper. The item also needed to be a solid color so that the camera could detect the object. Using the resources available to them, the team decided that the robot would attempt to pick up a mozzarella stick, carry it to its dunking location, where it was dipped in marinara sauce, and lifted off the field. During tests, the gripper had trouble grasping the mozzarella stick due to its weight. This could be improved in future tests by increasing the elastic or spring force that holds the gripper closed.

CONCLUSION

This project began with the construction of the robot arm. Each part was 3D printed and assembled. The control board / brain of the robot was also soldered. Lastly, the MATLAB / Linux workstation was set up on each computer.

The success of this project relied on the development of a computer vision algorithm capable of detecting colored balls. Within the algorithm, HSV masks, region props, and median filtering were used to locate balls on the field. Based on their color, the robotic arm would move over to the ball, pick it up, and deposit it in a unique location coordinating to its color. The computer vision ended up being a success due to the recalibration of HSV filters and extrinsic camera parameters each time the robot was used.

When programming the robot, an event driven state machine was used. The robot can be classified into four states:

IDLE, SCAN, MOVE, and GRASP. Using these states, the robot successfully progressed through the tasks of object detection and localization, dynamic object tracking, and unique object manipulation.

This project introduced the concepts of forward and inverse positional and velocity kinematics using DH Parameters and the manipulator jacobian. Using these results, functions were able to be written in MATLAB. This project also introduced software tools such as MATLAB, Linux, and git command line. These concepts introduced in this class encompass the three core divisions of robotics. The calculations for the forward and inverse positional and velocity kinematics encompass the concept of mechanical engineering. The circuit development encompasses electrical engineering, while the MATLAB software development utilizes the key concepts of computer science. The concepts used by this project strengthened skills relating to vision control, actuation, and manipulation that will be utilized in future projects and tasks.

APPENDIX A: AUTHORSHIP

Report	
Section	**Author**
Abstract	CJG
Intro	ALL1
Methodology	
Calibration	ALL1
New 3D Prints	CJG
Camera Calibration	ALL1
Image Processing	ALL2
State Machine	ALL1
Results	ALL
Discussion	
State Machine	ALL1
Ball Calculations	CJG
Image Processing	ALL2
New Robot Functions / Trajectory	ALL1
Conclusion	ALL2 + CJG
Video	ALL1 + CJG
Code	
Camera Calibration	ALL
Ball Position Kinematics	CJG
Image Processing	ALL2
Gripper Functions	CJG
Robot Ball Manipulation Functions	ALL1
State Machine	ALL1 + CJG
Final Arm Tweaking and Tuning	ALL + CJG
Part Design / 3D Printing	CJG + ALL1
Dynamic Object Tracking	CJG
Unique Object Detection	CJG + ALL1

Key

ALL = Everyone

ALL1 = Amber Lee Lindberg

ALL2 = Aaron Lee Longo

CJG = Casey Gosselin

APPENDIX B: KINEMATICS CALCULATIONS

Forward Kinematics Calculations

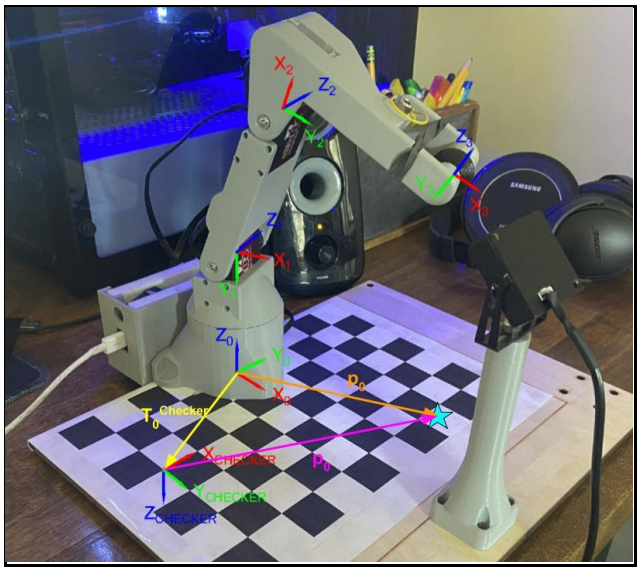

FIGURE 9: FIGURE SHOWING THE DH PARAMETERS AND FRAMES FOR THE ARM IN ZERO POSITION.

DH Parameters				
Joint	**θ**	**d**	**a**	**α**
1	q1	L0 + L1	0	-90
2	q2 - 90	0	L2	0
3	q3 + 90	0	L3	0

TABLE 1: THE DH PARAMETERS FOR THE ROBOT

$$p_0 - T_0^{Checker} * p_{checker}$$

$$T_0^{Tip} = \begin{matrix} 0 & 1 & 0 & 50 \\ 1 & 0 & 0 & -100 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

$$p_x = x\ position\ of\ end\ effector$$

$$p_y = y\ position\ of\ end\ effector$$

$$p_z = z\ position\ of\ end\ effector$$

$$q_1 = \theta_1 = \tan^{-1}\frac{p_y}{p_x}$$

$$q_2 = \theta_2 = 90 - \phi_1 - \phi_2$$

$$q_3 = \theta_3 = 90 - \phi_3$$

$$projection\ of\ arm\ onto\ xy\ plane = r = \sqrt{p_x^2 + p_y^2}$$

$$distance\ from\ joint\ 2\ to\ the\ tip = r_3 = \sqrt{r^2 + \left(p_z - (L_0 + L_1)\right)^2}$$

$$\phi_1 = cos^{-1}\frac{(L_3^2 - L_2^2 - r_3^2)}{-2L_2 r_3}$$

$$\phi_2 = \tan^{-1}\frac{p_z - (L_0 + L_1)}{r}$$

$$\phi_3 = \cos^{-1}\frac{r_3^2 - L_2^2 - L_3^2}{-2L_2 L_3}$$

Inverse Kinematics Calculations

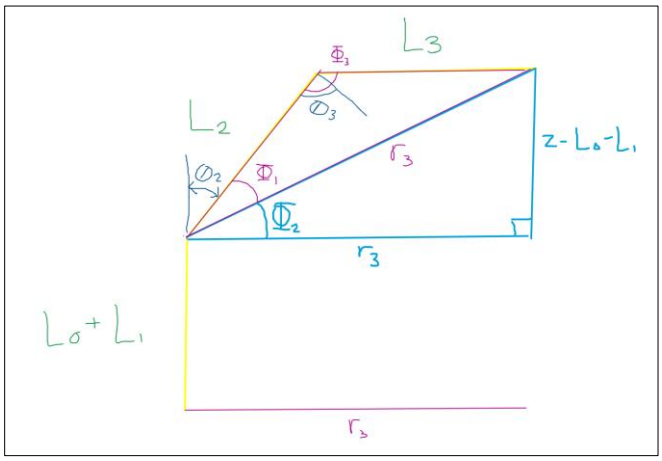

FIGURE 10: FIGURE SHOWING THE INVERSE KINEMATICS CALCULATIONS FOR THE ROBOT

APPENDIX C: GITHUB RELEASE LINK

Final Release: https://github.com/RBE300X-Lab/RBE3001Code09/tree/Final_Project

Extra Credit 1: https://github.com/RBE300X-Lab/RBE3001Code09/tree/Final_Project_EC1

Extra Credit 2: https://github.com/RBE300X-Lab/RBE3001Code09/tree/Final_Project_EC2

APPENDIX D: ARM DEMONSTRATION VIDEO

https://youtu.be/ZmY3_EOfiiY

APPENDIX E: BALL CENTROID DIAGRAMS

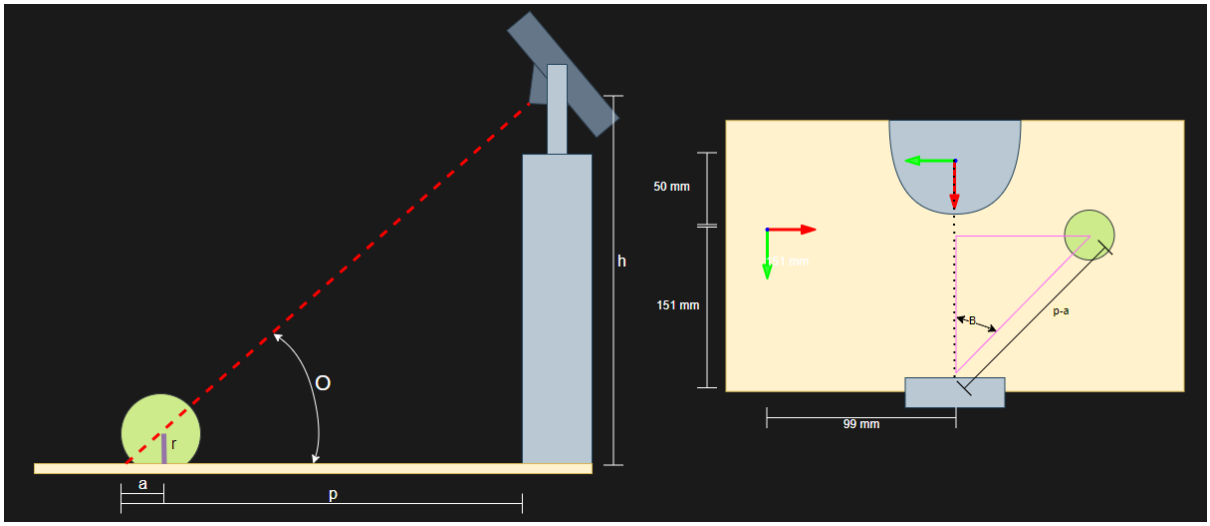

FIGURE 11: THE DIAGRAM USED FOR CALCULATING THE CENTER OF THE BALL.

APPENDIX F: STATE MACHINE DIAGRAM

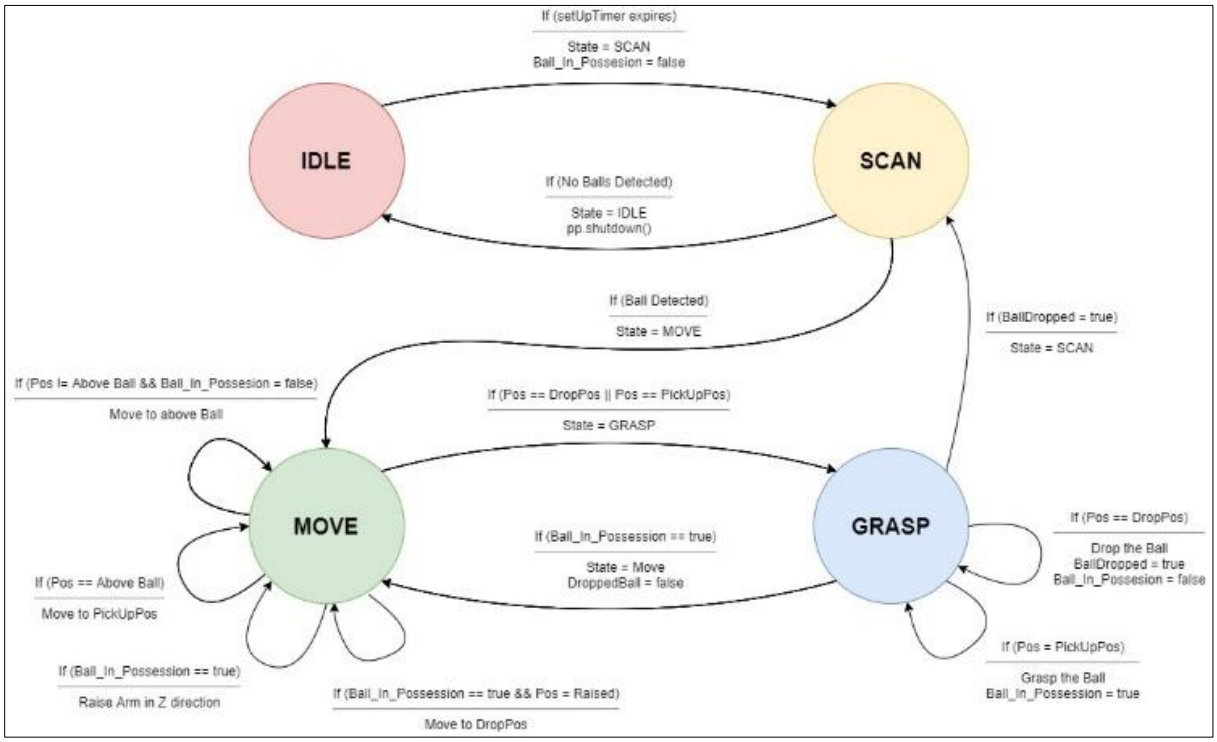

FIGURE 12: THE EVENT-DRIVEN STATE MACHINE USED BY THE ROBOT.

APPENDIX G: SYSTEM ARCHITECTURE DIAGRAM

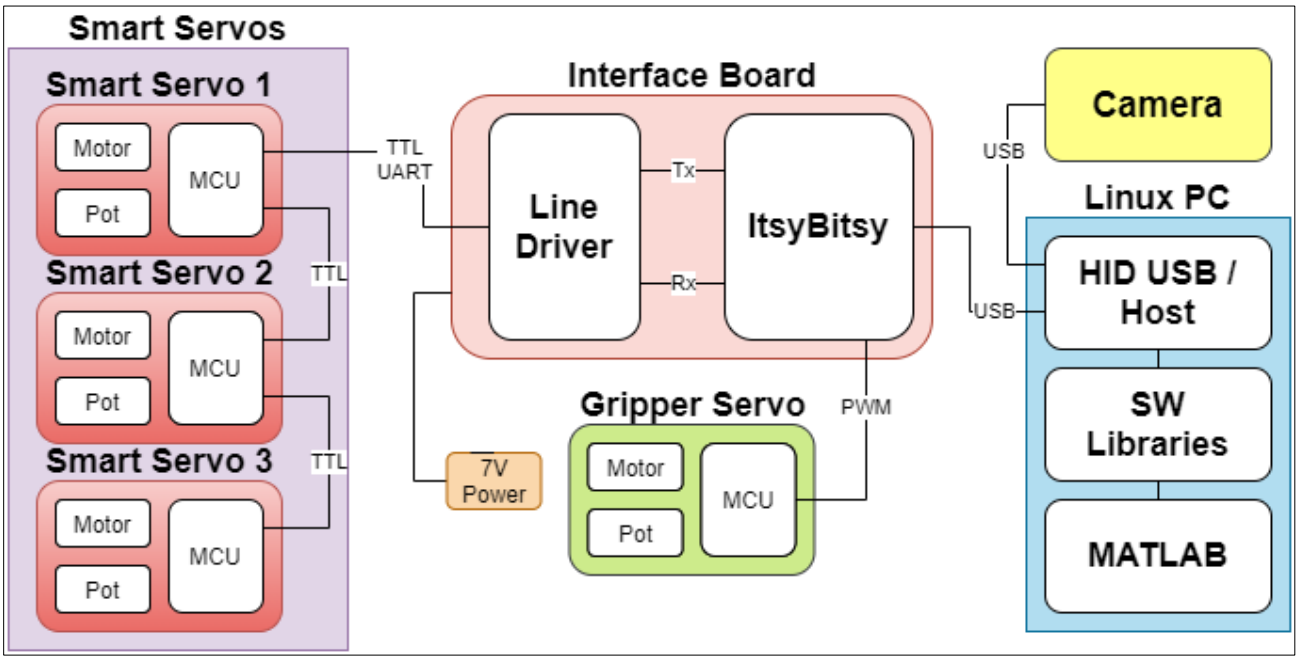

FIGURE 13: THE OVERALL SYSTEM ARCHITECTURE