Group 12: Ball Catching Robot

Sasank Potluri
potluri.sas@northeastern.edu
Northeastern University

Venkatesh Desai
desai.ven@northeastern.edu
Northeastern University

Vinesh Krishna Anne
anne.v@northeastern.edu
Northeastern University

Vasu Kumar Reddy Sirigiri
sirigiri.v@northeastern.edu
Northeastern University

Abstract

This project aimed to develop a system capable of tracking and touching a dynamically moving ball using a robotic arm guided by computer vision techniques. The primary challenge involved using a monocular camera, which provides only 2D image coordinates, necessitating the estimation of 3D coordinates for accurate trajectory prediction and robotic control. Initially, a color-based segmentation approach using HSV masks was explored for ball detection but proved less accurate under varying lighting conditions. Subsequently, the state-of-the-art YOLOv8 object detection model was employed, leveraging its robust performance in localizing and tracking the ball across video frames. The 3D coordinates of the detected ball were estimated using the 'cv2.solvePnP' function in OpenCV, and ArUco markers were strategically placed in the scene to establish the required poses and coordinate system alignments. By integrating these computer vision techniques with physics-based trajectory prediction models, the system demonstrated the capability to track and touch a ball rolling on the floor, showcasing the synergy between computer vision, trajectory estimation, and robotic manipulation for precise and dynamic interactions with moving objects.

Github- https://github.com/sasank98/ball_catching_robot

1 Introduction

In the field of robotics and automation, the ability to precisely manipulate objects in motion is a challenging and invaluable skill. This project aims to develop a system capable of catching a ball using a robotic arm guided by computer vision techniques. The successful execution of this task requires the integration of various disciplines, including image processing, 3D coordinate estimation, trajectory prediction, and robotic control.

The primary objective of this project is to design and implement a robust solution that can accurately detect and track a ball in projectile motion, estimate its 3D coordinates, predict its future trajectory, and control a robotic arm to intercept and catch the ball at the appropriate time and location. This aim has numerous potential applications, ranging from sports training and entertainment to industrial automation and material handling.

2 Implementation/ System Details

2.1 System Configuration

In the core of our ball-catching robot system lies the **Reactorx 200 robot arm**, which is integral to the operation. This robot arm is known for its precision and agility, making it ideal for applications requiring quick and accurate movements. The robot arm has 5 Degrees of Freedom, providing a wide range of motion. Works with python-ROS, Gazebo, and Rviz. The vision capabilities of our system are enhanced by a high-resolution **webcam** that plays a crucial role in the tasks of ball segmentation and tracking. This webcam records video in full color, which is vital for advanced image processing algorithms like YOLO-V8 and HSV to identify and track the ball's movement in real-time.

Camera Calibration: Camera calibration is the process of estimating the intrinsic and extrinsic parameters of a camera to establish the mapping between 3D world coordinates and 2D image coordinates.

To perform camera calibration, we captured images of a known calibration pattern (i.e. a chessboard) from different viewpoints and orientations to obtain accurate calibration results. The calibration pattern provides known 3D world coordinates and their corresponding 2D image coordinates.

We used functions like **cv2.findChessboardCorners**-to detect the chessboard corners in each calibration image, **cv2.cornerSubPix**-to refine the detected corner locations for better accuracy, use **cv2.calibrateCamera**-to estimate the intrinsic and extrinsic parameters based on the collected calibration data. We then saved the estimated intrinsic and extrinsic parameters for later use in our project.

Aruco Markers (Augmented Reality University of Cordoba): They are a type of fiducial marker system designed for camera pose estimation and augmented reality applications. They consist of a black border and an internal binary matrix that provides a unique identifier for each marker. ArUco markers are widely used in robotics, computer vision, and augmented reality due to their robustness, ease of detection, and accurate pose estimation capabilities.

We used the inbuilt OpenCV function **cv2.aruco.detectMarkers** to detect the Aruco markers present in the input image. This function returns the corners of the detected markers and their IDs. Once we had detected the markers, we used **cv2.aruco.estimatePoseSingleMarkers** or **cv2.aruco.estimatePoseBoard** (for multiple markers) to estimate the camera pose relative to each marker or the whole marker board. These functions require the detected marker corners, the marker IDs, and the known dimensions of the markers. The pose estimation functions return the rotation vectors and translation vectors that describe the position and orientation of the camera relative to the marker(s). To convert the 3x1 Rotation Matrix to the 3x3 Rotation Matrix we made use of **cv2.Rodrigues** to convert the rotation vectors to rotation matrices for easier interpretation and calculation of transformation matrices.

3 Methodology and Experiment

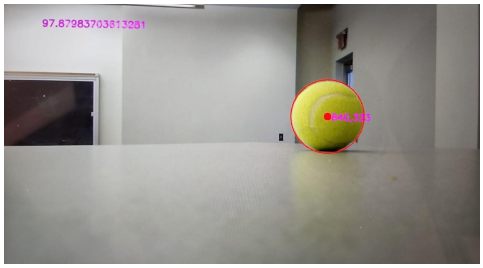
In this section we mainly discuss the details of our implementation, our project can mainly be divided into various parts: Ball localization, Calibration and transformation, and Trajectory estimation and Robot Planning.

3.1 Ball Localization Methods

We used two different methods to detect the ball, find the 3D coordinates of the ball, and find the Transformation Matrix of the ball with respect to the robot ${}^{B}T_{R}$. The HSV method is fast in ball detection but unreliable as it works well only in a similar environment which was used to detect the HSV range of the ball. YOLOv8 is a highly accurate and efficient object detection model, capable of detecting objects in real-time. It can handle varying scales, orientations, and occlusions of the ball.

(A) Manual Segmentation: For this method, we first converted the BGR color (as OpenCV operates on BGR by default) frame to the HSV color space using cv2.cvtColor(frame, cv2.COLOR_BGR2HSV). We did this after applying Gaussian blur on the BGR image to reduce noise in the image. We then determined the HSV range that best represents the color (Light Yellow color) of the ball we wanted to detect. We manually experiment with different HSV values on sample frames. We then created a Mask using cv2.inRange to create a binary mask where pixels within the specified HSV range are set to 1 (white), and the rest are set to 0 (black). Morphological operations like erosion and dilation were used to remove noise and fill in any gaps in the detected regions. We used cv2.findContours to find the contours (outlines) of the detected regions in the binary mask. The mask still had multiple contours, therefore our next task was to filter out the contours that did not match the expected size and shape of the ball. We used contour area, aspect ratio, or other geometric properties to identify the contour that best represents the ball. For the filtered contour representing the ball, we used functions cv2.moments and cv2.boundingRect to obtain the 2D coordinates (x, y) of the ball's centroid or bounding box in the image. Once we had the 2D image coordinates of the ball, we used the cv2.solvePnP function along with the camera calibration parameters to estimate the 3D coordinates of the ball in the camera's coordinate system.

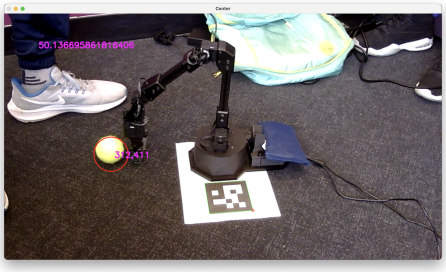
(a) Environment setup for Image (b)



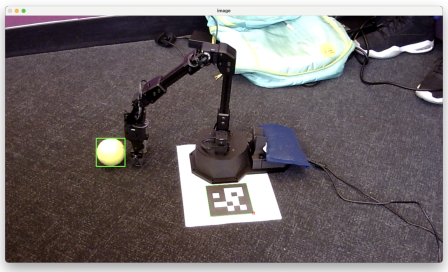
(b) HSV method shows a good accuracy under setup (a)

Figure 1: Environment setup and Ball segmentation

(B) Yolov8 detection: We use one of the pre-trained models provided by Ultralytics (yolov8m.pt, yolov8l.pt) We pass the frames captured by the camera to the model for detection of the ball. In the yolov8 model, we are interested in the "sports ball" class for the detection of our ball. The Model returns the bounding box of the ball from which the midpoints of each rectangle side are considered, these 4 points in image space would correspond to (-r, 0, 0), (0, -r, 0), (r, 0, 0) and (0, r, 0) in the real world. cv2.solvePnP is used with the above given 3D points and it's 2D correspondences in image space, which gives us the camera pose w.r.t ball center and the inverse of it would give us the ball's location in Camera coordinates. We made use of Mac M1 GPU for this task and it took 42ms to detect the ball in the frame.



(a) Ball detection using Manual Segmentation Method



(b) Ball detection using Yolov8 method

Figure 2: Ball detection using Manual Segmentation and Yolov8

3.2 Calibration and Transformation

Currently, our Ball's 3D location is in Camera co-ordinates and needs to be transformed to the Robot's co-ordinate system to perform the grasping operation. We used an Aruco tag to transform the ball from camera co-ordinates to Robot's co-ordinates. We placed the Aruco tag at a known distance from the origin of the Robot and made sure that it is visible to the camera. Once the program is turned on the first frame of the camera is used to compute the transformation matrix between the camera and the Robot and ball localization is performed in the rest of the frames. Using an Aruco tag to localize the ball, rather than the camera placed in the top would give us the flexibility to place the camera at any place and angle which would also help us throw the ball from any direction.

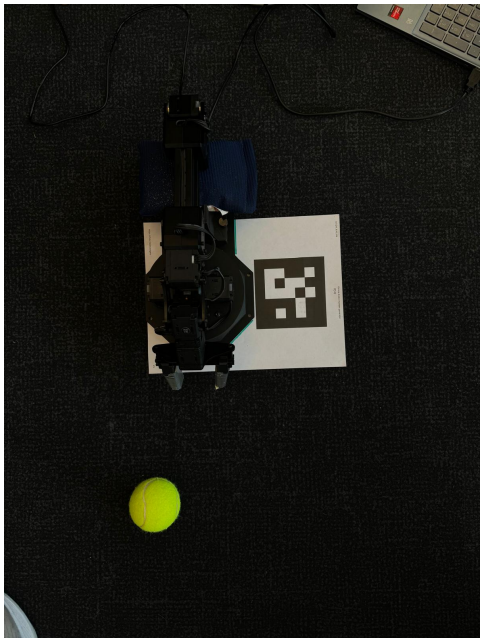


Figure 3: Setup to transform from Camera Co-ordinates to Robot Co-ordinates

3.3 Trajectory Estimation and Robot Planning

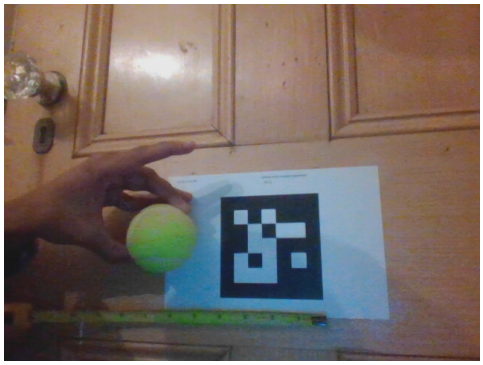
When a ball is rolled towards or near the robot we assume it doesn't have any spin and travels in a straight line. To give the arm more time to react we only take the second and third ball estimation values, while testing our project we found out often times our first reading of the ball is when it is detected in the edge of the image and the full size of the ball is not detected. The second and third readings of the ball worked the best during our testing. If the ball is rolled on the same plane as the Robot the Z-axis reading would be 3cm(radius of the ball for all the cases) and the end-effector can go to a lowest value of 1.7cm, beyond which the arm will hit the ground, due to these reasons we don't need the z-axis reading of ball location. Using the two sets of X and Y values we can interpolate a straight line, We assume a Radius of 0.25 meters for the Arm to reach and draw a circle. Using the straight line and circle calculated above we can compute co-ordinates of the points where the two curves intersect, from the co-ordinates we compute the angle our Robotic arm would have to rotate, and the absolute minimum angle is chosen from them. As soon as the first reading of the ball is received the robotic arm would move from its sleep position to a distance of 0.25 meters and a waist angle of zero degrees, after the angle is computed the waist joint is rotated by the computed angle. We specifically went with this strategy to avoid any problems with Joint limits and velocity limits

Figure 4: Experiment Setup

4 Results

(A) Ball Segmentation using Manual techniques: To have real-time speeds we first used a Manual Segmentation technique where we converted the image to Hue, Saturation, Value and then thresholded the image, which was then followed by post-processing techniques, this wasn't very accurate and often times needed extensive tuning based on the lighting and the background, which is why went with YOLOv8-L model to perform ball detection

(a) Manual Segmentation Mask for the environment shown in Manual Segmentation



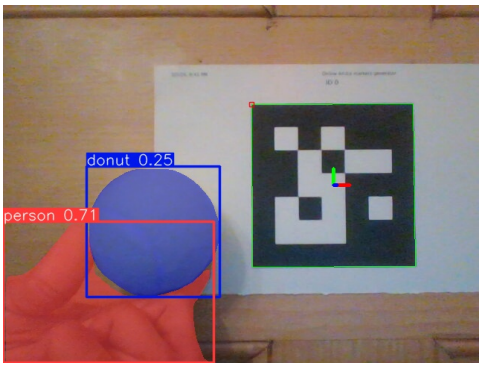
(b) Setup for calculating the accuracy

Figure 5: Results showing Manual Segmentation and our measurement technique

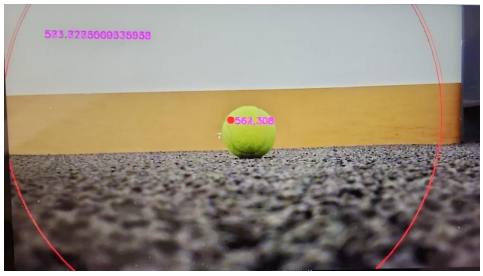
(B) Ball Accuracy calculation: One of the most crucial parts of this project is getting an accurate 3D location of the Ball. To measure the accuracy of the ball we first computed the ball location and measured the distance with tape from the camera, but in this case distance will be a scalar so later we started to use an Aruco tag and placed the ball at a known distance from the Aruco tag and computed the error between the known baseline distance and the same baseline computed by subtracting Aruco tag's position and the ball's position. In the above, given both cases, our error was under 3cm even at a distance of 1.5 meters from the camera

5 Challenges and Solutions

(A) Ball Segmentation Issues: One challenge we found was during the ball localization due to the camera limitations and in the initial use of the YOLO-v8 model which gave wrong detections in some cases. Also, we found an average error of 1 cm with respect to the Aruco Tag. To solve these problems, we implemented a segmentation based on the HSV space using the ball's unique color. Our idea is sampling the ball's position for more frames could lead to a better trajectory estimate but this didn't work very well as it required extensive parameter tuning and the radius of the ball was never as accurate as YOLO-v8's prediction



(a) Image showing Wrong detection from YOLO-v8



(b) Manual segmentation method fails when a similar range is detected

Figure 6: Ball detection using YOLO and Manual Segmentation method

(B) Perception tasks on a different Laptop: Our other challenge involved running YOLOv8 on a GPU-enabled device to obtain object coordinates for the robotic arm. However, our existing system lacked GPU support for arms control. Consequently, we executed the object detection task on a Mac M1 device and relayed the transformed 3D coordinates to the arms control system. This was achieved through socket communication between the two devices.

(C) Robot End-effector Positioning: In our initial approaches to controlling the arm, we thought we would just give the X, Y, Z co-ordinates of the ball's trajectory near the robot using **bot.arm.set_ee_pose_components**, but that kept giving us problems related to safety like joint limits exceeded or the velocity limits exceeded, later we tried using **bot.arm.set_ee_pose_matrix** method itself but it never worked that well, our assumption for the Robotic arm is if the Z-axis value is below +5cm the Joint and velocity limits would exceed. We even tried increasing the task time but that didn't help. What finally worked is we started to put the Robot in the pose we thought might have less angle on each Joint and started to do **bot.arm.get_ee_pose_matrix** and then asked it to go to the same position again. We found one configuration like that where the arm looks like it is approaching to catch the ball from the top and control the waist angle to coincide with the ball's direction.

6 Future Work

We could improve object detection by using TensorRT to optimize the YOLOv8 pre-trained model for improved performance on NVIDIA GPUs. TensorRT is a high-performance deep learning inference optimizer and runtime that can significantly accelerate the execution of deep learning models.

We could also use an object tracking algorithm like a Kalman filter or a correlation tracker to track the ball's position across consecutive frames, especially when the ball moves rapidly or is occluded.

Due to the unexpected challenges we came across during our project, we couldn't accomplish the final task of catching the ball in the projectile motion. Therefore, we would like to execute this part of the task after speeding up our detection algorithm and implementing object tracking in the project.

Figure 7: Robot End-effector Positioning

Our ball Position still has an error of 1-2cm, to get a better position estimate of the ball we could try and use Stereo or Multi-view Cameras.

7 References

[1] Opencv Documentation, https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

[2] Interbotix Documentation, https://docs.trossenrobotics.com/interbotix_xsarms_docs/

[3] ROS (Robotic Operating System) Documentation, https://wiki.ros.org/

[4] AprilTag Documentation, https://april.eecs.umich.edu/software/apriltag

[5] P. Cigliano, V. Lippiello, F. Ruggiero and B. Siciliano, "Robotic Ball Catching with an Eye-in-Hand Single-Camera System," in *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1657-1671, Sept. 2015, doi: 10.1109/TCST.2014.2380175.