Robotic Systems I Tic-Tac-Toe playing Robot

Yashas Karthik
Electronics and Communication
PES University
Bengaluru, Karnataka, India
yashaskarthikb@gmail.com

***Abstract*— This paper aims to demonstrate the procedure and methodology used to build and develop a code that runs a robot, specifically MyCobot 280 M5, to play tic tac toe along with. Since the robot wasn't very accurate, I have used 2 differently coloured blocks as the 'x's and 'o's, which will be picked up and placed using the robot end effector, suction pump.**

***Keywords*—colour recognition, cv2, camera, cobot, mask.**

I. Introduction

Smart robotics refers to the integration of artificial intelligence (AI) and advanced sensors into robotic systems, enabling them to operate autonomously, adapt to changing environments, and make intelligent decisions. These robots can perceive their surroundings, analyze information, and learn from experiences to improve their performance over time. Smart robotics find applications in various industries, including manufacturing, healthcare, agriculture, and logistics, where they can enhance efficiency, precision, and safety in tasks ranging from assembly and inspection to surgery and exploration. The combination of AI and robotics opens up new possibilities for automation and innovation, paving the way for a more interconnected and intelligent future.

The OpenCV (cv2) module in Python is a versatile library for computer vision tasks, encompassing a rich set of tools for image processing. When it comes to colour detection, cv2 enables the manipulation and analysis of images based on their colour content. This involves the conversion of images into various colour spaces, such as RGB (Red, Green, Blue) or HSV (Hue, Saturation, Value), and the application of filters or thresholding techniques to isolate specific colours or colour ranges. This functionality is invaluable in robotics for tasks like identifying objects based on colour, tracking them, or segmenting them from the background. ArUco markers, on the other hand, are specialized markers designed for augmented reality and robotics applications.

These markers typically feature a black-and-white pattern that is easily detectable by computer vision systems. In the realm of robotics, ArUco markers serve a crucial role in tasks such as pose estimation, localization, and navigation. Equipped with cameras, robots can identify these markers within their environment.

The unique patterns on ArUco markers allow robots to precisely determine their position and orientation relative to these markers. This information enhances the robot's spatial awareness, enabling it to navigate and interact with its surroundings more effectively.

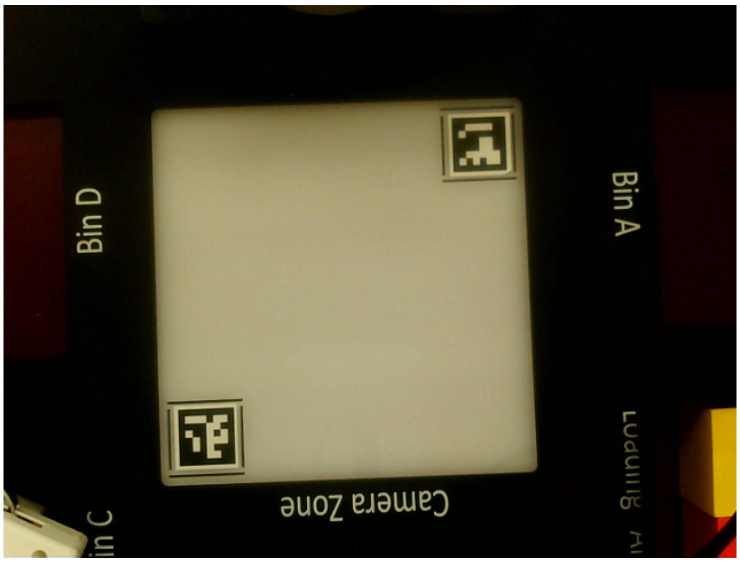
In essence, the cv2 module empowers developers and researchers to perform nuanced colour detection in images, which is crucial for various robotic applications. Simultaneously, ArUco markers serve as identifiable landmarks that robots leverage for accurate localization and spatial understanding, contributing to enhanced autonomy and functionality in robotic systems.

II. METHODOLOGY

1) The code starts with opening the camera to detect the arUco markers which are in front of the robot. so the robot knows the relative distance from the arUco board to the robot.

```
cv2.VideoCapture(cap_num,cv2.CAP_V4L)
```

2) Once the arUco stickers are detected it later uses cv2 and asks the user to crop out the output frame such that only the arUco board is seen in the cv2 frame.



3) This is done as a calibration test, such that the colour detection is done accurately.
4) This is also done such that we can only see the board where the blocks will be placed. So no other color in the camera's frame will be mistakenly detected as the block.
5) The code also connects to the robot and creates a robot object to communicate to the robot to make it move to the right position to pick up and place blocks.
6) The code remembers the cropping ratio and uses it in a while loop for every subsequent frame of the camera output.
7) Now that the video is cropped, we draw the actual tic tac toe board on the CV2 frame, so we know where we're placing the blocks. This will help differentiate where the block has been placed by the user. This is done by dividing the cv2 frame into 3 parts, by length and width which creates the 3x3 board.
8) We also initiate the board, which will be represented as a 3x3 list of empty strings, which

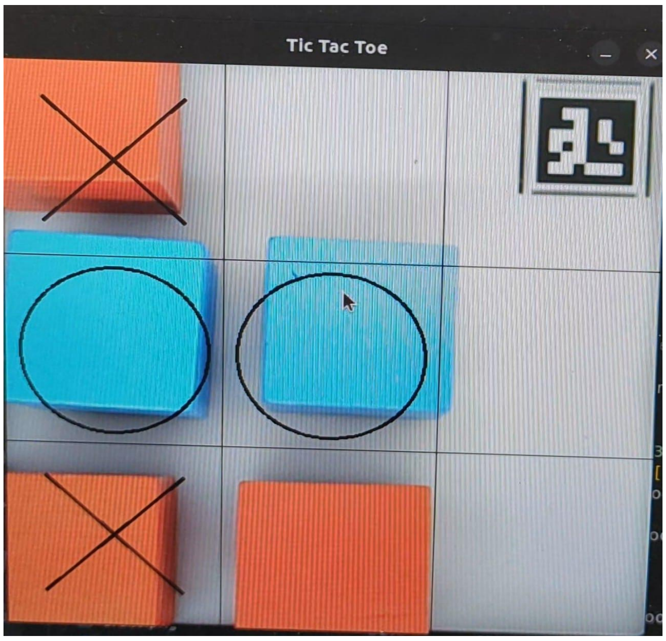
will be updated to either an X or O every time there is an input to the tic tac toe board.

```
[[' ' for _ in range(COLS)] for _ in
range(ROWS)]
```

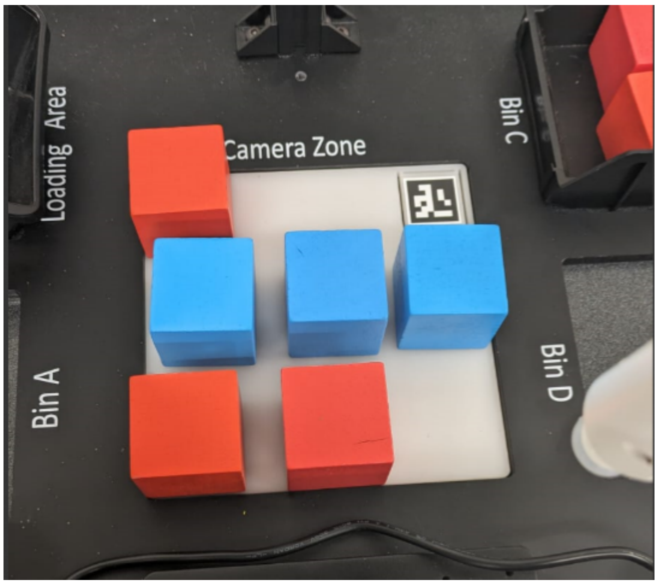
9) Since the camera was placed in the opposite direction of the robot, it was necessary to flip the image.
10) Once the camera is set up, it prompts the user to make their move and hits the spacebar on the keyboard to bring the robot notice that the move is made.
11) Once the spacebar has been pushed, the code takes a snapshot of the current frame. With this new frame, it searches for all the contours of that specific colour(red) and draws a rectangle around each of them.
12) Below are the upper and lower limits of the HSV for red, green and blue. using this it can detect either of those colors.

```
lower_green = np.array([35, 43, 35])
upper_green = np.array([90, 255, 255])

lower_blue = np.array([100, 43, 46])
upper_blue = np.array([130, 255, 255])

lower_red = np.array([0, 43, 46])
upper_red = np.array([10, 255, 255])
```

13) Now it calculates the center of all of these boxes and appends it to a list. It then checks if the centre of these rectangles lies within any of the tic-tac-toe blocks and stores that row, or column value in a variable.
14) It then updates that row and column of the board value with 'X' replacing the space.
15) Now the code has recorded the current state of the board and plans its next move, The code is meant to be defensive and plays moves to block possible wins for the user.
16) Based on the user's input it calculates its next move. It now goes to a constant location to pick up its block. Now that it knows where it has to place the block it turns on the suction picks up and places the block in the designated square.
17) The important thing to remember is the orientation of the end effector while picking up and placing the block, to ensure that it is placed properly in the board, we need to have the orientation to be the same.
18) Each iteration updates the board value with the respective X and O values and shows the same as a terminal output on the cv2 frame.



cv2 frame for the last move



Actual board after robot won the match

19) After each move the code checks if there are 2 x's or o's in a row or if the board is full to make sure it doesn't miss an ending.
20) Once one of those values are True, it print the corresponding text and exits the program.

III. OBSERVATIONS

The main inbuilt functions and modules used to perform this task were.

Modules:

```
import cv2
import numpy as np
import time
import serial
import serial.tools.list_ports
from pymycobot.mycobot import MyCobot
```

Inbuilt functions-

```
serial.tools.list_ports.comports()
cv2.aruco.Dictionary_get(cv2.aruco.DICT_6X6_250)
cv2.aruco.DetectorParameters_create()
send_angles([0.61, 45.87, -92.37, -41.3, 2.02, 9.58], 20)
cv2.resize(frame, (0, 0), fx=fx,fy=fy, interpolation = cv2.INTER_CUBIC)
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.aruco.detectMarkers(gray,self.aruco_dict, parameters=self.aruco_params)
self.mc.set_basic_output(2, 0)
cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
cv2.inRange(hsv,lower_color,upper_color)
cv2.findContours(color_mask,cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cv2.boundingRect(contour)
cv2.line(frame, (0, i * cell_height), (width, i*cell_height), (0, 0, 0), 1)
cv2.circle(frame, center, radius-5, (0, 0, 0), 2)
mc.send_coords([x, y, 103, 179.87, -3.78, -62.75], 40, 1)
frame = cv2.flip(frame, -1)
cv2.waitKey(1)
cv2.imshow("Tic Tac Toe", image)
cv2.VideoCapture(cap_num, cv2.CAP_V4L)
```

IV. RESULTS

Drive link for photo and videos

Youtube video of explanation