Competition Lab
Maze Search and Rescue robot

Group 2

Team Members:

Ratnesh Ravikiran Yadav

Sadjad Mahnan

Yue Zhu

Contents

1. Lab Summary

The objective of the competition lab was to design and develop an autonomous robot that could navigate a maze, locate, and "rescue" objects, simulating a search and rescue operation in a hazardous environment. This required us to integrate various technical skills and concepts, including robotic design, sensor integration, and algorithm development, to create a robot capable of autonomously exploring and transporting objects (tennis balls) from a maze setup.

We focused on several crucial aspects. We designed the Lego chassis to ensure it was capable of maneuvering through the pathways of the maze without getting stuck. We implemented a finite state machine (FSM) to manage the robot's behaviors effectively for wall following. The integration of sensing mechanisms was critical for detecting the walls of the maze and locating the objects within it.

The project taught us not only how to use image processing and finite state machines to build a robot but also how to implement them together successfully. The functionalities of the robot could have been improved by establishing a better communication protocol between Raspberry Pi and Arduino Uno to process all the data received.

2. Design

2.1. Hardware Design

2.1.1. Mechanical Design

Figure 1 shows the design of our robot. We decided to use LEGO bricks to construct the main components of our robot. This decision was driven by the versatility and modularity of LEGO bricks, allowing us to iterate on our design with flexibility and ease. The chassis of the robot was entirely built from LEGO blocks, providing a sturdy yet lightweight base that could be easily modified to accommodate other components or adjustments based on testing feedback. The ball collection mechanism was also designed using LEGO. It featured a simple yet effective method to "drag" tennis balls into itself.

For mobility, the robot was equipped with a total of six wheels: four passive 360º rotational mini ball casters allowed for smooth multidirectional movement, enhancing maneuverability within the maze’s variable confines. Additionally, two active drive wheels provided the necessary traction and power to navigate and overcome obstacles within the maze path. This combination of passive and active wheel systems enabled our robot to have enhanced control and stability.

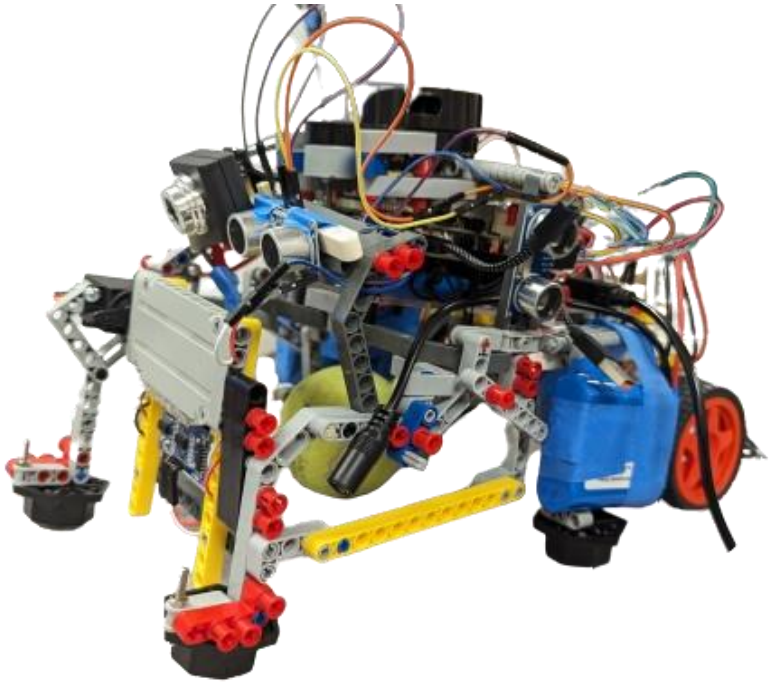

Figure 1: Wall Following and Ball Collecting Bot

2.1.2. Electrical Design

All the electrical components and their role in the robot are listed below:

- Raspberry Pi4 Model B: For onboard computation.
- Arduino Uno: To control sensors and motors.
- Webcam (Lab kit): To detect tennis balls for image processing.
- L298 DC Motor Driver: To drive high torque DC motors.
- 12V 1200 mAh Li-ion Battery: To provide power for the robot.
- 12V 10000 mAh Powerbank: To provide power for RPi 4
- M2596 Buck Converter: To drive 5V servo motor of ball collecting mechanism.
- MG996R Servo Motor: To drive the ball collecting mechanism.
- 12V DC High-speed Encoder Gearmotor: To drive the robot effortlessly.
- Ultrasonic Sensors: To detect the distance of the robot from the front, the left, and the right side.

2.2. Robot Behaviour and Control

The main finite state machine had three states, with the first state having another sub-finite state machine.

- S0: Wall Following (Left Wall Following)
  .1. S0: Follow The Wall
  .2. S1: Turn Left
  .3. S2: Turn around
  .4. S3: Stop
- S1: Ball Collecting
- S2: Go Back

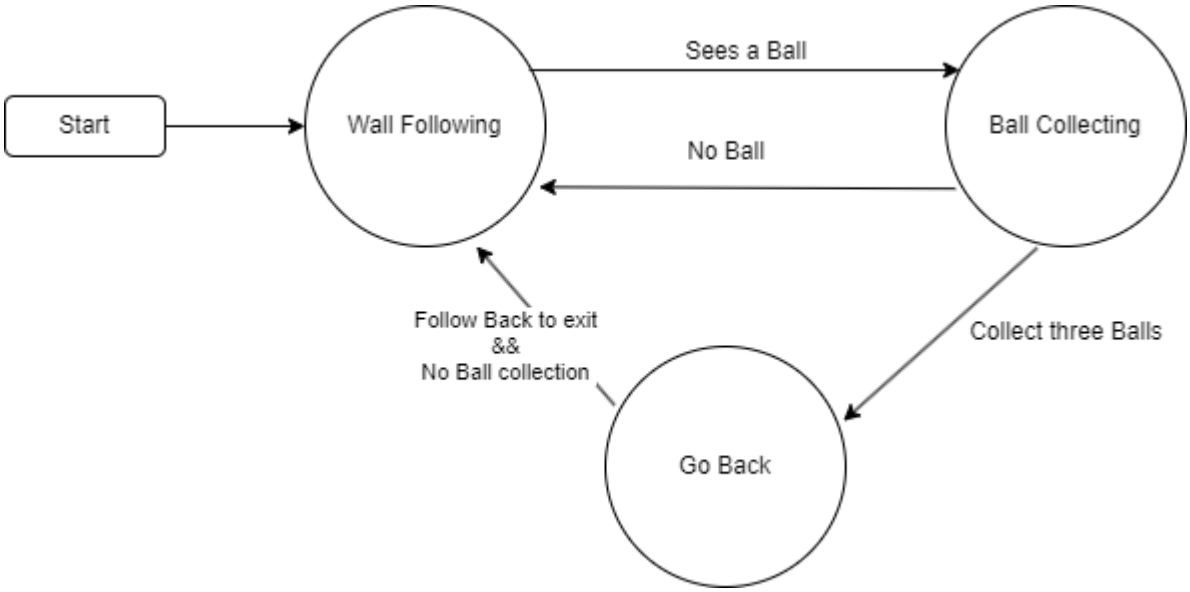

Figure 2: Main Finite State Machine

The robot would start at state S0. Now, if during this stage, at any point, it detects the ball, it would transition to state S1. Once at this state, the servo controlling the ball collecting mechanism would start up, and the robot would slow down and proceed toward the ball. Once it no longer sees the ball, it would transition back to state S0.

An additional sensor is placed to count the number of balls collected; if it reaches three, the robot will transition to state S0, no longer collect balls, and follow the walls toward the exit.

Figure 2 shows the main Finite State Machine.

During State S0, it will transition to its substate finite machine. At this stage, it will continuously monitor the distance between itself and the ball and take action. For example, if the distance detected by the left sensor crosses the threshold distance and the distance detected

by the front sensor is still below the threshold value, it would mean an opening on the left, causing the robot to move left. Figure 3 shows the sub-finite state Machine. All python scripts are included in Appendix A.

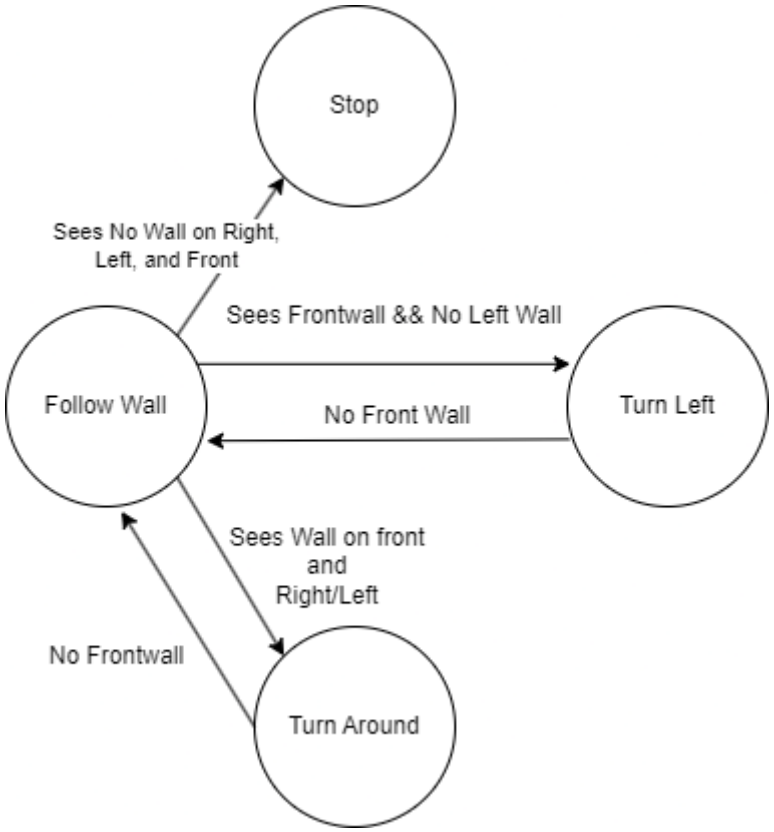

Figure 3: Wall Following Sub-Finite State Machine

2.3. Sensing Mechanism

2.3.1. Image Processing

The image processing was done using OpenCV and Python on Raspberry Pi 4. The image was first converted from RGB to HSV and then the green balls were detected using HSV thresholding. Once a circular contour with the largest contour area was detected, its position was sent to the Arduino.

2.3.2. Wall Distance Collection

The distance between walls on all sides was detected using the three ultrasonic sensors placed on the front, right, and left sides of the robot (Figure 4). The initial design also included a LIDAR sensor placed on top of the robot; however, due to the slow data reception speed, it was not incorporated in the final design. The full Arduino script is included in Appendix B.

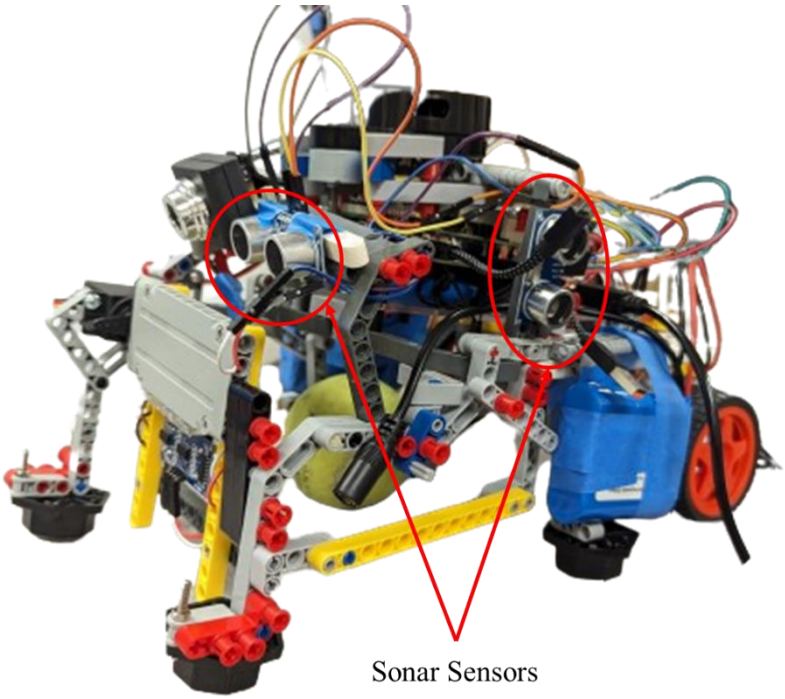

Figure 4: Ultrasonic (Sonar) Sensors

2.3.3. Ball Count Mechanism

An ultrasonic sensor was placed on the ball collector mechanism to keep track of the number of balls collected. Figure 5 shows the ball sensor counter.

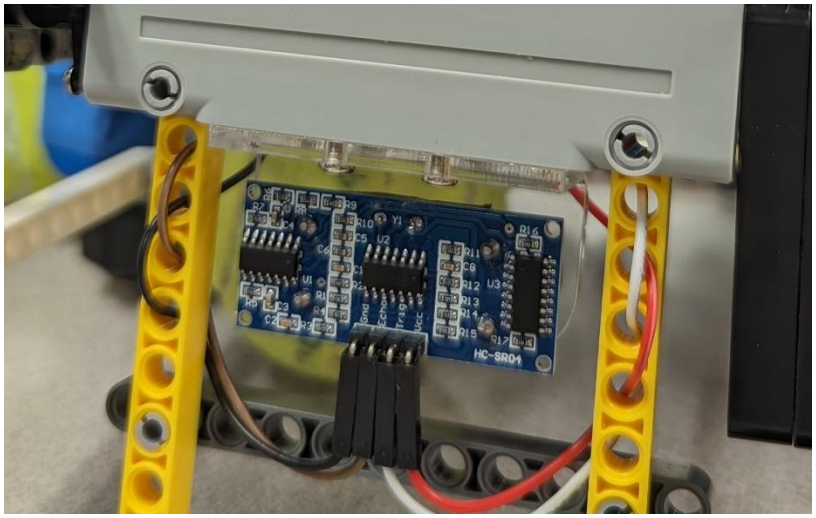

Figure 5: Ultrasonic Sensor Placement for Ball Count.

2.4. Ball Collection Mechanism

During State S1, when a ball is detected, the robot slows down, and the servo controlling the collection mechanism positions itself to let the door-like collector structure open up and then drag the ball within itself (Figure 6). Once the ball is collected, it reverts to State S0.

Overall, it was designed to collect three green balls and then head back to the exit. However, due to the uncertainty of sometimes being unable to get back the reflected signal from the ball, the mechanism would sometimes remain in state S1.

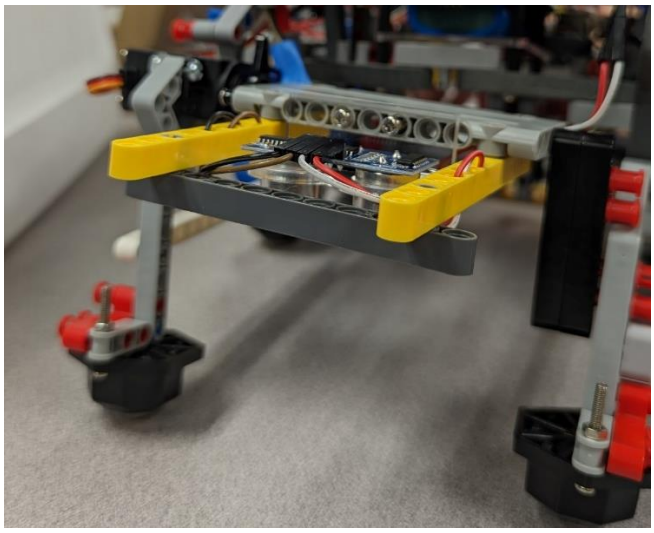

Figure 6: Ball Collector

2.5. Batteries and Power Management

Two 12V rechargeable 1200 mAh Li-ion batteries and a 10000 mAh power bank were used to power the entire bot. The two 12 DC motors were driven using one Li-ion battery via motor driver. The second Li-ion battery was used to drive the servo motor. The voltage level was first stepped down to 5V using an M2596 Buck converter. The power bank was used to power Rpi 4. Initially, we had different 5V power supply designated for Raspberry Pi. However, since we switched from RPi 3 To RPi 4, the 5V power supply was insufficient. As such, it was not incorporated, and a 12V power bank was used instead. Figure 7 shows the power supplies used.

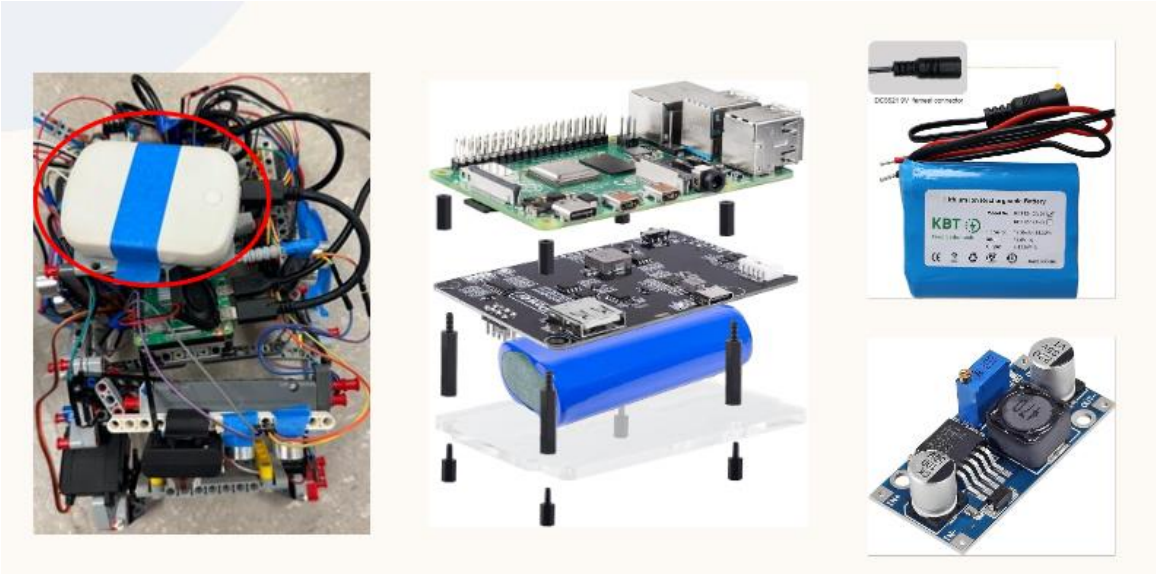

Figure 7: Power Supplies

2.6. Results

In general, the bot's performance fell short of expectations. It managed to follow walls efficiently and track the balls. However, we encountered a setback when the 7A motor driver we relied on during the competition burned out. A single 2A L298N motor driver proved insufficient to power both DC motors. Consequently, we had to utilize two of these motor drivers to achieve the desired performance.

3. Conclusion

All in all, after the design and development of the bot, we learned how to apply the skills taught in the class for the competition. The bot was able to follow the wall and find the tennis ball. However, owing to the technical issue of burning out the motor driver on our end, we could not run the bot on the day of the competition.

For the future work, the design of the robot could have been improved by adding the following functionalities:

- Using CAN serial communication instead of USB serial communication to transfer data between Arduino and Raspberry Pi smoothly.
- Using an alternate method to avoid uncertainty in the ball count mechanism.
- Using object detection to detect balls of any color.

Having said that, one of the issues we frequently faced while using the Raspberry Pi allotted to us was having a continuous flickering connection with the internet. So, getting this issue solved for the next group would be great.

Appendices

Appendix A

Main Computation Script (Raspberry Pi)

```
import cv2  # state of the art computer vision algorithms library

from lidar_funcs import open_close_cp2102
from arduino_com_func import connect_ard
from web_cam_funcs import init_get_frames, ball_tracking_v2
from strategy import ball_pick, PID_wall_following
import time

if __name__ == '__main__':
    # init, start program
    window_size = 3
    data_store_f = []
    data_store_l = []
    data_store_r = []
    forward_dist_1 = []
    left_dist_1 = []
    right_dist_1 = []
    uturn_threth = 500
    exit_threth = 1200

    # init cam
    img_width = 500
    img_height = 480
    block_pix = 80
    cap = init_get_frames.InitCam(img_width, img_height)
    if cap:
        print('cam opened')

    # init arduino
    ser_ard = []
    open_series = 1
    is_open, ser_ard = connect_ard.ConnArd(open_series, ser_ard)
    if is_open:
        print("Arduino port {} is open".format(ser_ard.name))
    else:
        print("Arduino port is not open")
```

```
    # wait for init
    time.sleep(2)

    # start loop
    '''
    Wall following (no ball found): follow the right wall, if front distance is small (wall a head), tank turn left.
    if front distance is not too large but the right distance is too large, fully turn right because of the door
    if the front distance is too large and the left distance is too large, stop
    Ball tracking (ball found): Follow and pick the ball up, if the door is open, just go straight until the door is closed
    '''
    ball_count = 0
    currenttime = time.time()
    force_wall_follow = 0
    while True:
        # get distances from Arduino by sonar
        while True:  # loop until get correct data
            try:
                sonar_ball_data = ser_ard.readline().decode().strip()
                if sonar_ball_data:
                    values = sonar_ball_data.split(',')
                    if len(values) == 4:
                        sonar_ball_data = [int(value) for value in values]
                        '''
                        Serial.print(sonar_r);
              Serial.print(",");
              Serial.print(sonar_l);
              Serial.print(",");
              Serial.print(sonar_f);
              Serial.print(",");
              Serial.println(ball_count);
                        '''
                        right_dist_1 = sonar_ball_data[0]
                        left_dist_1 = sonar_ball_data[1]
                        forward_dist_1 = sonar_ball_data[2]
                        ball_count = sonar_ball_data[3]
                        if forward_dist_1 != []:
                            forward_dist = forward_dist_1
                        if left_dist_1 != []:
                            left_dist = left_dist_1
                        if right_dist_1 != []:
```

```
                            right_dist = right_dist_1
                        break
            except:
                pass

        print('front: {}'.format(ball_count))
        print('left:{}'.format(left_dist))
        print('right:{}'.format(right_dist))

        # print(right_dist)
        # Wall Following, max_speed = 1


        turn_mode, PID_left_right =
PID_wall_following.LeftWallFollowing(distance_front=forward_dist,
                                                                    di
stance_left=left_dist,
                                                                    di
stance_right=right_dist,
                                                                    ut
urn_threth=uturn_threth,
                                                                    ex
it_threth=exit_threth)
        max_speed = 1  # full speed during wall following
        servo_pos = 0  # close door during wall following
        wall_follow = 1  # 1 wall following, 0 pick ball;

        if ball_count < 3:  # when collected 3 balls, only wall following
            # get track from web cam
            grabbed, BGRimage = init_get_frames.ObtainBGRFrame(cap)
            #cv2.imshow('BGRimage', BGRimage)
            if not grabbed:
                continue
            if_tracked, tracked_image, center_point =
ball_tracking_v2.BallTracking(BGRimage, block_pix)
            if if_tracked: # and min(forward_dist, left_dist, right_dist) >
150:
                print('ball tracked')
                #cv2.imshow('track ball', tracked_image)
                PID_left_right = ball_pick.TrackBall(center_point)  # [-1,1]
                percent_speed, servo_pos =
ball_pick.SpeedCloseBallandPick(center_point)
                max_speed = percent_speed
                wall_follow = 0
```

```
            elif servo_pos == 1:
                max_speed = 0.1  # move slowly to collect the ball
                turn_mode = 1
                PID_left_right = 0
                wall_follow = 0
            grabbed = 0
            if_tracked = 0
            del tracked_image
            del BGRimage

        if cv2.waitKey(1) == ord("q"):
            break

        '''
        data structure: servo position (0 init or 1 open or -1 dont care
from python);
        turn mode (-1 tank turn sig, 0 stop sig, 1 normal turn sig);
        turning control sig (PID: -1 left, 0 straight forward,1 right);(if
tank turn, -1 left, 1 right)
        max_speed (0 stop, 1 full speed))
        '''
        #trigger_yes = 0
        #while not trigger_yes:  # loop until get correct data
            #try:
              #  trigger_send = ser_ard.readline().decode().strip()
             #   trigger_yes = int(trigger_send)
            #except:
            #    pass
        # confirmation_data = "ACK"
        # ser_ard.write(confirmation_data.encode())

        # wait for arduino to confirm
        # response = ser_ard.read(len(confirmation_data)).decode()
        # if response == confirmation_data:
        data_to_send = "Y{};{};{};{};{}\n".format(str(servo_pos),
str(turn_mode), str(PID_left_right), str(max_speed),
                                                  str(wall_follow))
        ser_ard.write(data_to_send.encode())
        print(data_to_send.encode())
        time.sleep(0.1)  # wait for arduino to receive data

        # print(data_to_send)

    # close everything when down
```

```
    open_series = 0
    is_closed, ser_ard = connect_ard.ConnArd(open_series, ser_ard)
    if is_closed:
        print("Arduino port is closed")
    else:
        print("Arduino port is not closed")
    cap.release()
```

Webcam

```
from web_cam_funcs import init_get_frames, ball_tracking_v2
import cv2
img_width = 640
img_height = 480
cap = init_get_frames.InitCam(img_width, img_height)
while True:
    grabbed, BGRimage = init_get_frames.ObtainBGRFrame(cap)
    if not grabbed:
        break
    if_tracked, tracked_image, center_point =
ball_tracking_v2.BallTracking(BGRimage)
    if if_tracked:
        cv2.imshow('tracked_image', tracked_image)
        print(center_point)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Ball Tracking

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np


def BallTracking(image, block_pix):
    thre_area = 2500
    height, width, _ = image.shape
    # image = image[80:height, :]
    # convert bgr to hsv
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
```

```
    # Define color ranges for tennis balls
    colors = {
        # 'pink': ([162, 151, 188], [177, 224, 254]),
        # 'yellow': ([29, 200, 141], [34, 234, 234]),
        # 'orange': ([4, 153, 157], [7, 247, 236]),
        # 'red': ([0, 234,  58], [4, 255, 151]),
        # 'green': ([59, 186, 122], [66, 252, 198]),
        'real green': ([30, 114, 92], [50, 233, 255])
    }


    # Create masks and apply morphology operations
    masks = {}
    for color, (lower, upper) in colors.items():
        mask = cv2.inRange(hsv, np.array(lower), np.array(upper))
        # kernel = np.ones((15, 15), np.uint8)
        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15, 15))
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
        masks[color] = mask
        # cv2.imshow('1', mask)

    # Search for circular contours in each mask and stop after finding one
    largest_contour = None
    largest_contour_area = 0
    for color, mask in masks.items():
        contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        if contours:
            for contour in contours:
                area = cv2.contourArea(contour)
                if area > largest_contour_area and area > thre_area:
                    largest_contour_area = area
                    largest_contour = contour
        if largest_contour_area > thre_area:
            print(largest_contour_area)
            break
        else:
            largest_contour = None

    # Draw bounding circle around the largest contour if found
    if largest_contour is not None:
        ((x, y), radius) = cv2.minEnclosingCircle(largest_contour)
        center = (int(x), int(y))
```

```
        radius = int(radius)
        cv2.circle(image, center, radius, (0, 255, 0), 2)
        if center[1] > block_pix:
            return True, image, center

    # If no circular contours are found, return False
    return False, None, None
```

Ball Collecting

```
from strategy import PID_controller
'''
>390 very close, open door, very slow
>300 close, slow down
'''
Kp = 10.0
Ki = 0.5
Kd = 5.00
setpoint_distance = 500 // 2  # target distance pixel
pid_controller = PID_controller.PIDController(Kp, Ki, Kd, setpoint_distance)


def TrackBall(ball_location):
    '''
    :param ball_location: x,y
    :return: PID_output (how much left or right)
    '''
    ball_x = ball_location[0]
    PID_output = pid_controller.update(ball_x)
    # limit to [-1,1]
    PID_output = max(-2000, min(2000, PID_output)) / 2000
    return PID_output


def SpeedCloseBallandPick(ball_location):
    '''
    :param ball_dis_center: in mm
    :param ball_location: x,y
    :return: percent_speed, servo_pos to arduino
    '''
    if ball_location[1] > 350:  # pix
        percent_speed = 0.3
        servo_pos = 1 # close to ball and open door
```

```
    elif ball_location[1] > 300:
        percent_speed = 0.5
        servo_pos = 0
    else:
        percent_speed = 1
        servo_pos = 0
    return percent_speed, servo_pos
```

PID Controller

```
class PIDController:
    def __init__(self, Kp, Ki, Kd, setpoint):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.error = 0
        self.integral = 0
        self.derivative = 0
        self.last_error = 0

    def update(self, measured_value):
        self.error = self.setpoint - measured_value
        self.integral += self.error
        self.derivative = self.error - self.last_error
        self.last_error = self.error

        # PID output
        output = self.Kp * self.error + self.Ki * self.integral + self.Kd *
self.derivative
        return -output # flipped
```

Wall Following

```
from strategy import PID_controller

Kp = 15.0
Ki = 0.1
Kd = 5.00
setpoint_distance = 300# target distance in mm
pid_controller = PID_controller.PIDController(Kp, Ki, Kd, setpoint_distance)
```

```
def LeftWallFollowing(distance_front, distance_left, distance_right,
uturn_threth, exit_threth):
    # # -1 tank turn sig, 0 stop sig, 1 normal turn sig
    # # during turning, -1 left, 0 straight forward,1 right
    # print(distance_front, distance_left, distance_right)
    if distance_left > exit_threth and distance_front > exit_threth and
distance_right > exit_threth:  # if distance front and left are too far,
left the maze mm
        # stop
        # print('stop')
        return 0, 0
    elif distance_left > uturn_threth and distance_front < exit_threth:  #
if the right distance increase suddenly and the front distance is not too
far, there is a door   mm
        # fully turn left
        # print('fully turn right')
        return 1, -1
    elif distance_front < 150:  # if too close to front, make tank turn to
right mm
        # tank turn left
        # print('tank turn right')
        return -1, 1



    else:  # wall following
        # print('wall following')
        control = pid_controller.update(distance_left)  # pid control

        # print(control, distance_right)
        control = max(-2000, min(2000, control)) / 2000
    # print(control)
    return 1, control
```

Appendix B

Arduino Script

```
#include "functions.h"

// define flags
int FLAG_PRE_BALL = 0;  // 0 ball is not covered by door, 1 ball is covered
by door

// init receive from Python
int servo_pos_py;
int turn_mode_py;
double PID_left_right_py;
double max_speed_py;
int wall_follow_py;

// servo motor
Servo servo996;  // servo object representing the MG 996R servo
int servo_degree;

// ball count
int ball_count = 3;
int pre_ball_count = 0;

// sonar distance
int sonar_r = 0;
int sonar_l = 0;
int sonar_f = 0;

void setup() {
  Serial.begin(9600);
  pinMode(start_pin, INPUT);
  pinMode(mot_pwm_r, OUTPUT);
  pinMode(mot_for_r, OUTPUT);
  pinMode(mot_rev_r, OUTPUT);
  pinMode(mot_pwm_l, OUTPUT);
  pinMode(mot_for_l, OUTPUT);
  pinMode(mot_rev_l, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(trig_rlf, OUTPUT);
  pinMode(echo_r, INPUT);
```

```
  pinMode(echo_l, INPUT);
  pinMode(echo_f, INPUT);

  servo996.attach(ser_mot);
  servo_degree = ServoControl(SERVO_INIT, servo996);
}

void loop() {
  // get sonar distances
  sonar_r = SonarDist(trig_rlf, echo_r);
  sonar_l = SonarDist(trig_rlf, echo_l);
  sonar_f = SonarDist(trig_rlf, echo_f);

  // send the distances to python for decision making
  Serial.print(sonar_r);
  Serial.print(",");
  Serial.print(sonar_l);
  Serial.print(",");
  Serial.print(sonar_f);
  Serial.print(",");
  Serial.println(ball_count);
  Serial.flush();
  delay(50);
  // FlushReadData();

//  int trigger = 1;
//  Serial.println(trigger);
//  // FlushReadData();
//  delay(50);

  // receive data from python
  bool dataIsValid = false;  // This flag checks the validity of the data

  // Loop until valid data is received
  while (!dataIsValid) {
    if (Serial.available() > 0) {
      String receivedData = Serial.readStringUntil('\n');
      Serial.flush();  // Clear the serial buffer to ensure fresh read next
time

      // Check if the data starts with the expected symbol
      if (receivedData.startsWith("Y")) {
        // Remove the start symbol for further processing
        receivedData.remove(0, 1);  // Remove the first character
```

```
        // Validate and parse the data
        if (validateData(receivedData)) {
          parseData(receivedData);
          dataIsValid = true;  // Set the flag to true when data is valid
        } else {
          Serial.println("Error: Data is incorrect or incomplete.");
          delay(500);  // Optional: delay before next read attempt
        }
      } else {
        Serial.println("Error: Data does not start with the correct
symbol.");
        delay(500);  // Optional: delay before next read attempt
      }
    }
    delay(10);  // Small delay to prevent overwhelming the CPU
  }

  if (servo_degree == SERVO_INIT){
    if (wall_follow_py) {
      // motor control for wall following
      // servo_degree = CloseDoor(servo996);
      if (turn_mode_py == 1) {  // normal turn
        MotorControlPID(PID_left_right_py, max_speed_py);
      } else if (turn_mode_py == 0) {  // stop
        MotorControlPID(PID_left_right_py, 0);
      } else if (turn_mode_py == -1) {  // tank turn
        MotorControlTankTurn(int(PID_left_right_py));
      }
    } else if (FLAG_PRE_BALL) {
      MotorControlPID(PID_left_right_py, max_speed_py);
    } else if (!wall_follow_py) {
      // motor and servo control for ball pick
      MotorControlPID(PID_left_right_py, max_speed_py);
      if (servo_pos_py == 1) {
        servo_degree = ServoControl(SERVO_ROT, servo996); // wrong value
from raspberry pi
        servo_degree = SERVO_ROT;
        FLAG_PRE_BALL = 1;
      }
    }
  }else if (servo_degree == SERVO_ROT) { // close door after the door is
open
      servo_degree = CloseDoorBySonar(servo996);
```

```
      MotorControlPID(0, 0.3);
      servo_degree = CloseDoorBySonar(servo996);
    if (servo_degree == SERVO_INIT) {
      ball_count++;
      FLAG_PRE_BALL = 0;
      MotorControlPID(0, 0);
      delay(1000);
    }

  }
  //FlushReadData();
  delay(10);
  // Serial.println(ball_count);
}

bool validateData(String data) {
  // Implement validation logic here, e.g., checking for number of
delimiters
  return (data.length() > 0 && data.indexOf(';') != -1);
}

void parseData(String data) {
  char str[100];
  data.toCharArray(str, 100);
  char* p = strtok(str, ";");
  int params[5];
  int i = 0;
  while (p != NULL && i < 5) {
    params[i++] = atof(p);
    p = strtok(NULL, ";");
  }

  if (i == 5) {
    servo_pos_py = params[0];
    turn_mode_py = params[1];
    PID_left_right_py = params[2];
    max_speed_py = params[3];
    wall_follow_py = params[4];
    Serial.println("Data successfully parsed and validated.");
  } else {
    Serial.println("Error: Incomplete data received.");
  }
}
```

```
void performActionsBasedOnData() {
  // Add your action code here
  Serial.println("Performing actions based on received data.");
}
```

Header Files

Header File 1

```
#ifndef _Functions_h
#define _Functions_h

#include <Servo.h>
#include <Arduino.h>
#include "strategy.h"

// DC motor pin
#define mot_pwm_r 3
#define mot_rev_r 4
#define mot_for_r 2
#define mot_pwm_l 6
#define mot_for_l 7
#define mot_rev_l 5

// servo motor pin
#define ser_mot 11

// DC motor
#define MAX_SPEED 135
#define MAX_SPEED_L 135


// servo motor
#define SERVO_INIT 180
#define SERVO_ROT 90

// sonar sensor
#define trigPin A0
#define echoPin A1
#define trig_rlf A2
#define echo_r A5
#define echo_l A3
```

```
#define echo_f A4

// start button pin
#define start_pin 2


// function def
void Mot_r(int dir, int speed);
void Mot_l(int dir, int speed);
void MotorControlTankTurn(int dir);
void MotorControlPID(double PID_output, double speed_ratio);
float SonarSensor(int turn_on);
int SonarDist(int trigger_pin, int echo_pin);
void WallFollow(int threth_distance);
int ServoControl(int degree, Servo servo);
void AnalyzeSerialData_v2(String receivedData, int& servo_pos_py, int&
turn_mode_py, double& PID_left_right_py, double& max_speed_py, int&
wall_follow_py);
void FlushReadData();
void AnalyzeSerialData(String receivedData, int& forwardDist, int& leftDist,
int& backDist, int& rightDist, int& ballCenterX, int& ballCenterY, int&
ballDisCenter, int& xCenterCoor);


#endif
```

Header File 2

```
#ifndef _Strategies_h
#define _Strategies_h
#include "functions.h"
#define SONOR_THRE_BALL 50
#define SONAR_THRE_BALL_WEIRD 600
#define SONAR_THRE_BALL_WEIRD2 80
#define SONAR_THRE_BALL_WEIRD3 100

int CloseDoorBySonar(Servo servo);
int OpenDoor(Servo servo);
int CloseDoor(Servo servo);
#endif
```

CPP Files

CPP File 1

```
#include "functions.h"

void Mot_r(int dir, int speed) {
  // 1 forward, 0 stop, -1 reverse
  if (dir == 1) {
    digitalWrite(mot_for_r, HIGH);
    digitalWrite(mot_rev_r, LOW);
    analogWrite(mot_pwm_r, speed);
  } else if (dir == 0) {
    digitalWrite(mot_for_r, LOW);
    digitalWrite(mot_rev_r, LOW);
    analogWrite(mot_pwm_r, speed);
  } else if (dir == -1) {
    digitalWrite(mot_for_r, LOW);
    digitalWrite(mot_rev_r, HIGH);
    analogWrite(mot_pwm_r, speed);
  }
}

void Mot_l(int dir, int speed) {
  // 1 forward, 0 stop, -1 reverse
  if (dir == 1) {
    digitalWrite(mot_for_l, HIGH);
    digitalWrite(mot_rev_l, LOW);
    analogWrite(mot_pwm_l, speed);
  } else if (dir == 0) {
    digitalWrite(mot_for_l, LOW);
    digitalWrite(mot_rev_l, LOW);
    analogWrite(mot_pwm_l, speed);
  } else if (dir == -1) {
    digitalWrite(mot_for_l, LOW);
    digitalWrite(mot_rev_l, HIGH);
    analogWrite(mot_pwm_l, speed);
  }
}

void MotorControlTankTurn(int dir) {
  // 0 stop, 1 right, -1 left
  if (dir == 1) {
    Mot_l(1, MAX_SPEED_L);
    Mot_r(-1, MAX_SPEED);
  } else if (dir == -1) {
    Mot_l(-1, MAX_SPEED_L);
    Mot_r(1, MAX_SPEED);
```

```
  } else {
    Mot_l(0, 0);
    Mot_r(0, 0);
  }
}

void MotorControlPID(double PID_output, double speed_ratio) {
  // control by PID
  int max_speed_loc = MAX_SPEED * speed_ratio;
  int max_speed_loc_l = MAX_SPEED_L * speed_ratio;

  if (PID_output >= 0) {
    // turn right
    int speed = max_speed_loc * (1.0 - PID_output);
    Mot_r(1, speed);
    Mot_l(1, max_speed_loc_l);
  } else if (PID_output < 0) {
    int speed = max_speed_loc_l * (1.0 + PID_output);
    // turn left
    Mot_r(1, max_speed_loc);
    Mot_l(1, speed);
  }
}

float SonarSensor(int turn_on) {
  if (turn_on) {
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    int duration = pulseIn(echoPin, HIGH);
    float distance = (duration * .343) / 2;
    delay(50);
    return distance;
  } else {
    digitalWrite(trigPin, LOW);
    return 0;
  }
}

int SonarDist(int trigger_pin, int echo_pin) {
  digitalWrite(trigger_pin, LOW);
```

```
  delayMicroseconds(2);
  digitalWrite(trigger_pin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigger_pin, LOW);

  int duration = pulseIn(echo_pin, HIGH);
  int distance = (duration * .343) / 2;
  delay(100);
  return distance;  // mm
}

int ServoControl(int degree, Servo servo) {
  servo.write(degree);
  return degree;
}

/*
    int servo_pos_py
    int turn_mode_py
    float PID_left_right_py
    float max_speed_py
*/

void AnalyzeSerialData_v2(String receivedData, int& servo_pos_py, int&
turn_mode_py, double& PID_left_right_py, double& max_speed_py, int&
wall_follow_py) {
  servo_pos_py = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);

  turn_mode_py = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);

  PID_left_right_py = receivedData.substring(0,
receivedData.indexOf(';')).toFloat();
  receivedData.remove(0, receivedData.indexOf(';') + 1);

  max_speed_py = receivedData.substring(0,
receivedData.indexOf(';')).toFloat();
  receivedData.remove(0, receivedData.indexOf(';') + 1);

  wall_follow_py = receivedData.toInt();
  delay(20);
```

```
}
void AnalyzeSerialData(String receivedData, int& forwardDist, int& leftDist,
int& backDist, int& rightDist, int& ballCenterX, int& ballCenterY, int&
ballDisCenter, int& xCenterCoor) {
  int partialData = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);  // remove analyzed
part
  if (partialData != -1) {
    forwardDist = partialData;
  }
  partialData = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);
  if (partialData != -1) {
    leftDist = partialData;
  }
  partialData = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);
  if (partialData != -1) {
    backDist = partialData;
  }
  partialData = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);
  if (partialData != -1) {
    rightDist = partialData;
  }
  ballCenterX = receivedData.substring(0,
receivedData.indexOf(',')).toInt();
  receivedData.remove(0, receivedData.indexOf(',') + 1);
  ballCenterY = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);
  ballDisCenter = receivedData.substring(0,
receivedData.indexOf(';')).toInt();
  receivedData.remove(0, receivedData.indexOf(';') + 1);
  xCenterCoor = receivedData.toInt();
}

void FlushReadData() {
  while (Serial.read() >= 0) {}
}
```

CPP File 2

```
#include "strategy.h"

int CloseDoorBySonar(Servo servo) {
  float distance = SonarSensor(1);
 Serial.print("Sonar:");
 Serial.println(distance);
  if (distance <= SONOR_THRE_BALL ) { // || (distance <=
SONAR_THRE_BALL_WEIRD3 && distance >= SONAR_THRE_BALL_WEIRD2   distance <=
SONOR_THRE_BALL
    delay(10);
    int servo_degree = ServoControl(SERVO_INIT, servo);
    distance = SonarSensor(0);
    delay(10);
    return SERVO_INIT;
  } else {
    return SERVO_ROT;  //signal not close the door
  }
}

int OpenDoor(Servo servo) {
  int servo_degree = ServoControl(SERVO_ROT, servo);
  delay(10);
  float distance = SonarSensor(0);
  return SERVO_ROT;
}

int CloseDoor(Servo servo) {
  int servo_degree = ServoControl(SERVO_INIT, servo);
  delay(10);
  float distance = SonarSensor(0);
  return SERVO_INIT;
}
```