

COE521- Embedded Systems

Section 31 – Fall 2021

Project – Tennis Ball Collector

Group 1

Hasan Basbous

Antoine Doumit

Roudy Abou Zeid

Omar Al Sayed

Contents

[Table of figures 2](#_Toc90877320)

[Components Used: 3](#_Toc90877321)

[Met requirements: 3](#_Toc90877322)

[Design: 4](#_Toc90877323)

[i. Rover Box Design 4](#_Toc90877324)

[ii. Picking Hand Design 5](#_Toc90877325)

[Difficulties 6](#_Toc90877326)

[Code Explanation 7](#_Toc90877327)

[Code of the PIC: 7](#_Toc90877328)

[Python Code[1] [3]: 8](#_Toc90877329)

[Conclusion 15](#_Toc90877330)

[References 15](#_Toc90877331)

# Table of figures

[Figure 1: Rover box design 4](#_Toc90877336)

[Figure 2: Ball picking hand design 5](#_Toc90877337)

[Figure 3: Robot whole design( includes second last version of the hand) 6](#_Toc90877338)

[Figure 4: Ports' configuration 8](#_Toc90877339)

[Figure 5: State "Locate Ball" 8](#_Toc90877340)

[Figure 6: Code Part I 9](#_Toc90877341)

[Figure 7: Code part II 9](#_Toc90877342)

[Figure 8: Ball color mask/contour 10](#_Toc90877343)

[Figure 9: Code part III 10](#_Toc90877344)

[Figure 10: Code part IV 11](#_Toc90877345)

[Figure 11: Added green area to the black box 12](#_Toc90877346)

[Figure 12: Code part V 13](#_Toc90877347)

[Figure 13: Code part VI 13](#_Toc90877348)

[Figure 14: Code part VII 14](#_Toc90877349)

[Figure 15: Code part VIII 14](#_Toc90877350)

# 

# Components Used:

To deliver the demo, we used the following components:

* Breadboard + wires
* 10K resistor
* 10MHz crystal oscillator + 2x20pf capacitors
* 5V power bank to power the pi
* Power adapter 5V 2.1A
* Rover 5
* Motor driver for the Rover
* PIC18F4550 and Pic Kit 3
* Raspberry Pi 3
* Raspberry Pi Camera

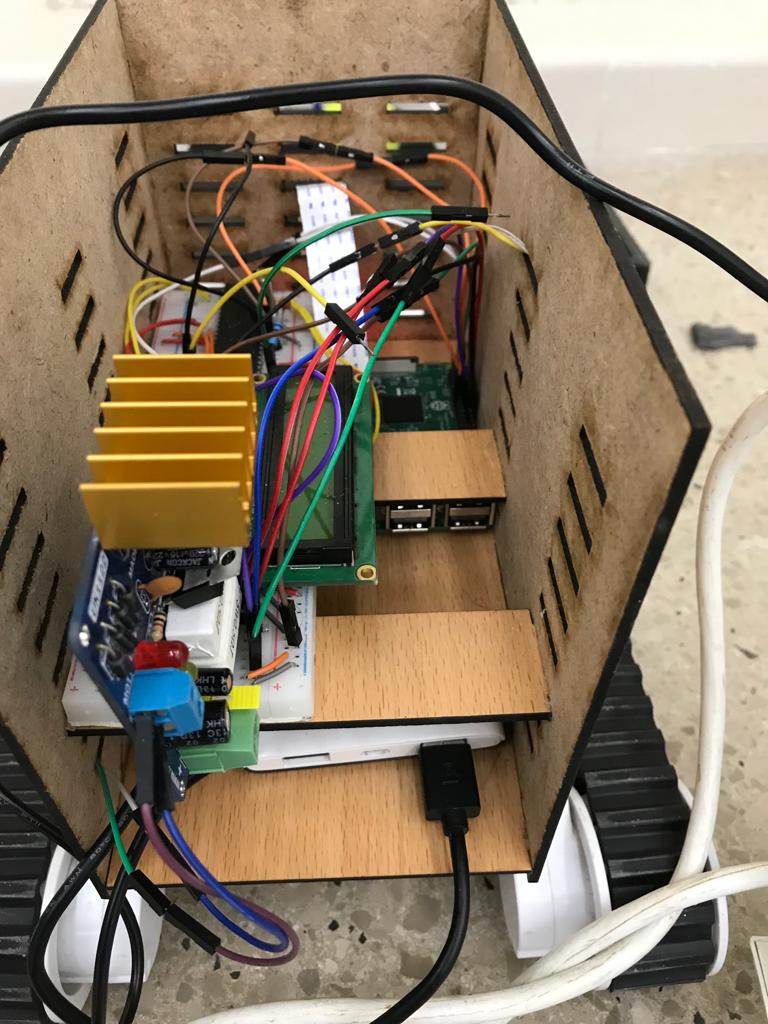
# Met requirements:

The robot was able to:

* Locate Balls in the white area
* Collect Balls
* Locate Box
* Deposit Balls
* Distinguish the location of the white area where the balls are and the black box( added a green color to the box) where to deposit the balls
* Successfully traverse the path from the initial point to the location of the balls and then to the black box
* Move more than one ball to the black box and sometimes all the balls

# Design:

## Rover Box Design



Power bank

Pi camera

Breadboard with LCD & PIC

Raspberry pi

Figure : Rover box design

To hold, the raspberry pi, raspberry pi adapter, breadboard with pic and LCD, motor driver, rover power adapter was a challenge.   
The first compaction made was using a power bank instead of the power adapter to power the raspberry pi.   
The second was to place the motor driver, LCD and pic on the same breadboard. We used short wiring on the breadboard, and only used the long wires between the pic and the raspberry.   
We placed the power bank at the bottom, on top of it comes the raspberry pi, and to the top of the raspberry pi comes the breadboard, in a way it keeps the pi pins uncovered. (The breadboard should be on the top to be able to view the LCD)

## Picking Hand Design



Support Edges

Blade

Ball trapper

Figure : Ball picking hand design

Axis

The hand is made of tough card box that is light weight, durable and cheap.

The hand passed through multiple design changes, the last is presented in figure 2. The idea of the hand is its dependency on the power of the rover motors to pick and drop the balls. When passing over a ball, the latter raises the hand (that is free to move upwards) which lets the ball in and 5cmx22cm flat card traps it from escaping.

To drop the ball, the hand should raise the ball 5cm, the height of the box. This is done by the aid of the support (refer to figure 2 for annotations), which will slide over the box wall upon the movement of the rover. This will raise the hand and the ball gradually as the rover moves forward towards the box. This assumes the box wall is tough enough to handle the rover pushing forward on it. (Note that the hand is free to move up to 7cm).

The hand has some degrees of freedom to the right and to the left, given by the axis it’s connected to with the rover. This is to make the hand adjust when the rover approaches the drop box at an angle.   
If we assume that the hand can’t move right or left, and the rover approaches the box at an angle, one of the side supports will rise more than the other, and cause the ball to reside on the other lower side, resulting in the ball being lower than the box wall, and a failure in drop.   
Thus, giving the hand some freedom to move right or left, will help ensure that both supports slide on the box wall at the same time with acceptable height difference between the supports.   
Note that if the angle with the box is too wide, the hand will fail, since we can’t have that freedom in the hand, else in normal moving of the robot, the hand won’t be straight.   
The initial design was made for the hand to move freely right and left, but when the rover moved right to pick the ball, the hand moved left and missed the ball. So, we decided to limit this movement to an extent where it’s still helpful in the case when it needs to drop the ball when the rover approaches the box at an angle.

A picture containing text, floor, indoor

Description automatically generated

Ball picking hand

Operations Box

Power cord

Figure : Robot whole design( includes second last version of the hand)

# Difficulties

* Making a compact design to place the power bank, raspberry pi, breadboard, LCD, pic and motor driver in one box.
* The most difficult was the design of the hand. The design passed through a minimal of four prototypes till an acceptable result was reached. Even the final one was still changed a little through out the testing.
* The results at the beginning of each testing session where positive, but a lot of testing seemed to affect the power of the rover. After two to three hours of testing, the power of the rover declined as we noticed it wasn’t able anymore to push the ball into the box, even with the same hand adjustments used at the beginning.
* Finding a code that serves our goal of detecting and following the ball, and applying changes to detect more the one ball, a box and perform certain behaviors like stopping at a specified distance.
* Detecting the black color is a problem, since a shadow in the room or a dark pants, or shoes was perceived as black color and thus we were forced to add another distinctive color, green. The other option was to switch to object detection instead of color detection, but we went too far with the color detection code that we didn’t have the time to revert to another one.

# Code Explanation

In this demo our code is divided into two: One code related to the PIC and the other one related to the Raspberry PI.

## Code of the PIC:

The pic in this project is given the part of interfacing the LCD to display the current state of the robot.

RA0-RA2 are used respectively as the RS, RW and EN of the LCD. PORTD is used as the data register of the LCD.   
RE2-RE0 are used to receive the state number from the raspberry, that will be displayed on the LCD.   
The A/D converter module is disabled, which makes all port pins digital. This is important for the LCD to function properly.

|  |  |  |  |
| --- | --- | --- | --- |
| RE2 | RE1 | RE0 | State |
| 0 | 0 | 0 | Locate Ball |
| 0 | 0 | 1 | Collect Ball |
| 0 | 1 | 0 | Locate Box |
| 0 | 1 | 1 | Deposit Ball |

We did not include a state for locating white area, since our rover is able to detect the balls even from a distance. Also, the white color is a challenge itself given that all the walls and a lot of objects are in white, this will add an unknown sudden behavior to the robot.

The figures below show the most important of the code of the pic, which is the ports’ configuration, and how one of the states is sent to the LCD. It’s important that at each state, the LCD is cleared and cursor set to initial position. The pic is made to check port E pins each 1s for a state change.

Graphical user interface, text, application

Description automatically generated

Figure : Ports' configuration

Text

Description automatically generated

Figure : State "Locate Ball"

## Python Code[1] [3]:

For this demo, the most two important python libraries we will use are OpenCV (cv2) and GPIO Zero (gpiozero) [2].  
Open CV is a library in python used in computer vision. GPIO Zero is a Python library for working with GPIO pins of the raspberry pi. If offers some helpful classes like Robot which has forward, backward, left, right and stop commands.

Text

Description automatically generated

Figure : Code Part I

The first part of the code specifies the camera resolution width and height, framerate, and the rover motors to which pins are connected. In this case left motor to GPIOs 25 & 8 and right motor to GPIOs 23 & 24. The gpiozero has a robot class that can control the movement of the rover through specifying the pins the motors are connected to, forward speed and turn speed.   
In this project, HSV (Hue, Saturation, Value) color representation is used instead of RGB, since the former is more robust towards external lighting changes when working with color detection.

The initial hue value is specified to be 30, which is the similar to the yellow in the tennis balls. To transform the image into HSV, there are lower and higher bounds where the HSV values can vary. In this case a margin of 10 is given around the hue value.

Text

Description automatically generated

Figure : Code part II

The second part of the code (referencing figure 5) specifies the pins that are going to send the states’ bits to the pic to be displayed on the LCD. These pins are not connected to LEDs, but for the sake of using gpiozero, the LCD class serves the point of setting the value of the pins to either 0 or 1.

Inside the for loop, a frame will be captured at a rate of 31 fps, which means a frame each 1/31 s. This frame is captured in bgr or RGB color representation, then transformed into HSV representation.   
A color mask or contour is created for pixels that have hsv values between the lower and upper bounds specified, as appears in the below figure.

Graphical user interface

Description automatically generated with medium confidence

Figure : Ball color mask/contour

Text

Description automatically generated

Figure : Code part III

For each contour of the capture contours (referencing figure 9), it’s bounded by a rectangle to be able to calculate its area and specify its center. The area will be the width \* height of the rectangle.   
We noticed that the center of coordinates is considered the top left corner with positive Y-values going to the bottom. Thus, the center of the contour will be the top-left corner + width/2 for x-value and the top-left corner + height/2 for y-value. The object with the biggest area contour is selected, and its area and center are saved in a ball\_location array holding [area, x-value, y-value].

Text

Description automatically generated

Figure : Code part IV

(Referencing figure 10)   
Initially we had specified a minimum and maximum area, where the minimum will tell the rover whether the object is found or to keep searching, and the maximum will tell it that whether it reached the object or not.   
  
How will the rover move towards the object?

The rover will turn around to the left as long as the captured frame has contours with area < than the minimum specified.

In case the area of the object falls between the minimum and the maximum, it means that the object is detected.   
The movement towards the object depends on if the center of the image frame, pixel (320, 240), aligns with the center of the detected object. If this is the case, the rover will move forward. Else if the center of the image frame + some margin value is smaller than the location of the center pixel of the object, the rover should move to the right. Else if the center of the image frame – some margin value is greater than the location of the center pixel of the object, then the rover should move to the left.   
Here we are interested in the x-value of the image center pixel and the object center pixel. The margin is the image width divided by a sensitivity variable. If we want the rover to correct more its direction towards the ball or the box, we increase the sensitivity. But this comes at an expense, where if we have multiple balls besides each other, it won’t be able to move since it will lock in a correction loop between the balls. Thus, the sensitivity should be adequate to avoid this case.

For how long it will move forward, to the left or to the right?

Each 1/32 secs, a frame is evaluated and accordingly the correction continues, until finally reaching a specified distance from the object.

The code is divided into multiple states controlled with count and count1 variables. The first run will be searching for the ball. The rover will turn in its place until it finds a ball, then moves towards it, till this point the state on the lcd will be “Locate Ball”. When the area reaches a certain value, the state is changed to “Collect Ball” since we are sure now that the rover found the ball (refer to the below figure).

Text

Description automatically generated with medium confidence

When the rover collects the ball, or when the area of the ball is bigger than the maximum\_value specified initially for the object area, this means that the ball is inside the hand, and the state is moved into “Locate Box”. Now the target is the box, so the color bounds with areas should be changed to fit the new object to be detected.

To be able to rerun the same code of getting the contours, detecting the largest of them, then correcting route, we change the lower and upper bound hsv colors to detect green area, which is added at the center of the box collecting the tennis balls as shown in the below image.

A picture containing text, green

Description automatically generated

Figure : Added green area to the black box

After the ball is collected the following is applied:

* Change the lower and upper bound to detect green color
* Change sensitivity that suits the object to be detected
* Change the minimum and maximum area, where the latter is specified in a way the rover stops away from at a distance of 15-20 cm.
* Change the state to ‘Locate Box’

Text

Description automatically generated

Figure : Code part V

This count value = 2 is used when the box is detected to execute the following code (code part VI) which changes the state to “Deposit Ball”. The rover being at a 15-20cm distance from the box is ordered to go in full speed for 2.5 seconds to ensure enough time and power for the hand to raise and drop the ball.

Text

Description automatically generated

Figure : Code part VI

The count1 value will track the number of balls detected and collected so far. This to ensure the rover goes back and search for other balls. In the case of code in figure 12, the rover will stop after depositing the fifth ball. After each deposit of a ball, there is a certain movement done by the rover, which is reverse for 3s then turn 180 degrees to the left; this to ensure that it doesn’t detect the balls in the box, and search for the balls in the white area which will be closer. Also, the hsv values are reset to that of the tennis ball, with count1 incremented, count and flag reset.

Text

Description automatically generated

Figure : Code part VII

The flag value mentioned above is used to change sensitivity or state when a certain distance from the object is achieved. Like in the below code, the first “if” is when the rover is at a distance of approximately 40cm from the ball, or when the area of the rectangle bounding the ball contour is greater than 2000. In this case, the state is changed to ‘Collect Ball’ and flag is incremented to not visit this if again. Similarly, in the second if, where the rover is at a distance where the area of the green area attached to the box appears to be 4000, the sensitivity is increased for a better alignment with the center of the green area.

Text

Description automatically generated

Figure : Code part VIII

# Conclusion

The project itself was a challenge as we should find a design of the hand given constraints of the box and the fact that the objects are rounded, so can’t be dragged like in the case of a box. At start, our goals were just to detect the ball and the box, then elevated to collecting the balls without a way to drop them in the box. After the first design of the hand, we raised the ceiling to catch and drop a ball. With more testing, we finally set the goal of catching all the five balls. Even though not all the times were fully successful, but it was appealing to see it work. The most challenging in the project is the time it needed for testing and dealing with uncalculated issues that arise.   
Every added line of code, and every added part on the hand and design has a story.

# References

[1] Code source (Matt-Timmons Brown): <https://github.com/the-raspberry-pi-guy/raspirobots>

[2] <https://gpiozero.readthedocs.io/en/stable/recipes.html#robot>

[3] <https://automaticaddison.com/how-to-make-an-object-tracking-robot-using-raspberry-pi/>

Online tools we used to transform RGB to HSV and to pick HSV values: <https://alloyui.com/examples/color-picker/hsv.html>

<https://www.rapidtables.com/convert/color/rgb-to-hsv.html>