The Luna T-Ball Robot

Tennis Ball Recognizing and fetching Robot

Robotic course Final Project

1. Introduction

This document specifies the "Luna Tennis Ball Recognizing Robot" project specification, in accordance to the Robotic course final assignment requirement.

1.1. **Purpose:**

- Most of the dog owners are facing this common problem, we do not have enough time to play with them and as a result, in many cases, the dog usually remains frustrated, which could lead to behavioral problems.
- Those problems could be in a form of a house and yard corruption, and in some other cases, rare ones, it could be in form of aggression.
- The Robot's main goal is to amuse a pet dog that likes to play with tennis balls. Dog owner who their dogs likes to play with tennis balls, not always have the power to play with their dogs and would rather someone else to do so. This is where the 'Luna TBall' Robot takes its place.
- The 'Luna TBall' Robot shall be able to recognize a tennis ball in its surrounding, drive over to it and stop right next to it.

1.2. **Not in the scope of this project:**

- In the future, the robot will be able to get the ball and throw it away for the dog to fetch

2. Project tools and hardware

2.1. **Hardware**

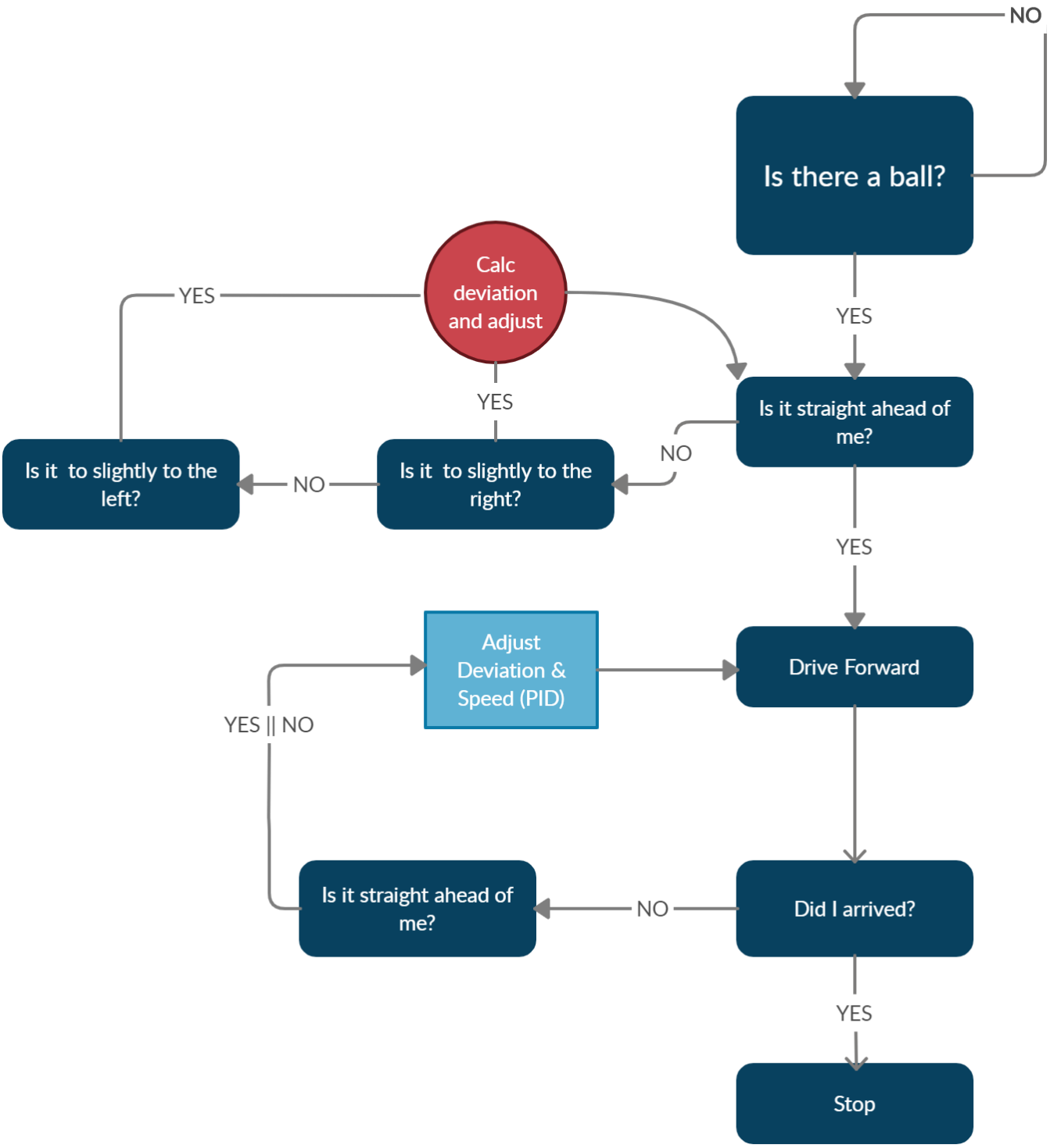
- Raspberry Pi 4 8GB – 60$
- Robot HATS shield for Raspberry Pi TS0737 – 14.79$
- PCA9685 PWM driver – 12.99$
- Motor driver module – 9.99$
- X3 Servo – 7.89$
- X2 DC Motors & Car Chassis – 35.99$
- X2 18750 3.7 Ion Batteries – 15.79$
- USB web camera – 6$
- Wires, resistors, bread board and other general components – 15$

Total amount(approximately) - 168.45$

2.2. **Software**

- Python 3.8 – Robot control
- Linux – Raspberry Pi configuration

3. Block diagram

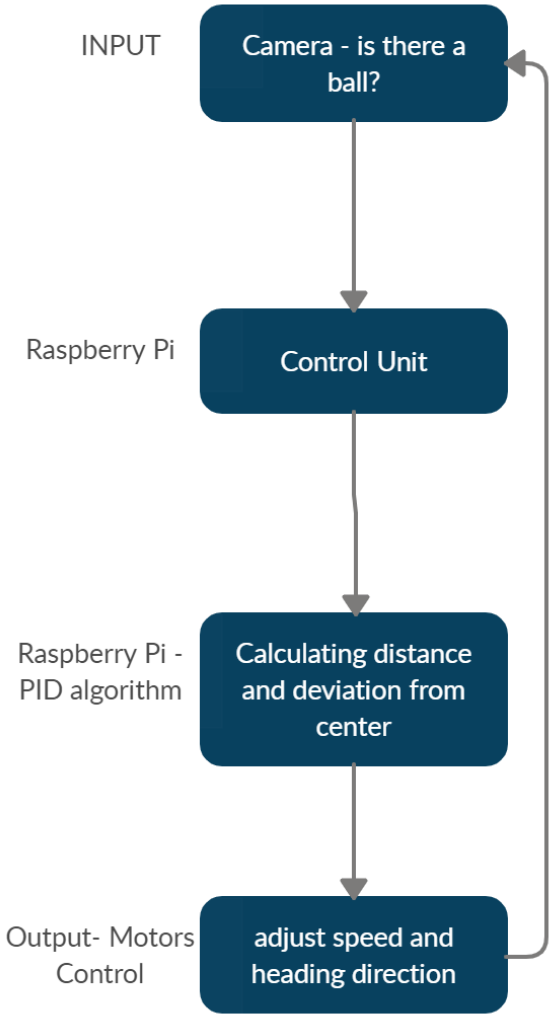


4. Course material that are implemented in this project

4.1. PID controller:

- In this project, I tried to write my own controller based on what we learned about PIDs controllers.
- By calculating the size of the ball in relation to the picture's frame size and to the object's location in the frame.
- The algorithm of the image processing is calculating constantly if there a ball, how far is it from the Robot and its location in the frame.
- As the Robot approaches to the ball and the size of its radius increases, it will slow down until it stops.
- If the ball is not centered in its field of view, it will adjust its position so it will be straight ahead of it.

4.2. PID flow diagram



4.3. Calculations general idea in the algorithm

- Frame size = W*H
- Ball min size – The robot arrived to the ball = H/3
- Ball max size – There is no ball if it is smaller than that = H/10
- Center of the Screen (X) = W/2
- Center of the Screen (Y) = H/2
- Ball size – determined by its radius calculation = Radius
- Center of the ball = center in (x,y) of the ball
- Speed

$$speed = \frac{ball\_size}{ball\_\min\_size} * constant$$

**If speed = constant, the Robot stops*

- Turning (left or right)

$$\Delta = screen\_center(X) - ball\_center(x)$$

**Positive Δ - steer right*
**Negative Δ - steer left*
**Δ = 0 – do not steer*

5. Resources:

5.1. OpenCV tutorial page for Python image processing

https://docs.opencv.org/3.4/index.html

5.2. Numpy tutorial page for python

https://numpy.org/

5.3. Picar library for controlling DC and servo motor

https://buildmedia.readthedocs.org/media/pdf/picar/latest/picar.pdf

5.4. Raspberry Pi tutorial

https://www.raspberrypi.org/documentation/

6. The Code for the whole robot (including image processing) – Python 3.8

```
from picar import front_wheels, back_wheels

from picar.SunFounder_PCA9685 import Servo

import picar

from time import sleep

import cv2

import numpy as np

import picar

import os


picar.setup()

# Show image captured by camera, True to turn on, you will need #DISPLAY and it also slows the speed of tracking

show_image_enable   = True

draw_circle_enable  = True

scan_enable         = True

rear_wheels_enable  = True

front_wheels_enable = True

pan_tilt_enable     = True


if (show_image_enable or draw_circle_enable) and "DISPLAY" not in os.environ:
                    #If there is no display connected will get this error

   print('Warning: Display not found, turn off "show_image_enable" and "draw_circle_enable"')

   show_image_enable   = True
```

```
   draw_circle_enable  = True


kernel = np.ones((5,5),np.uint8)
img = cv2.VideoCapture(-1)


SCREEN_WIDTH = 160
SCREEN_HIGHT = 120
img.set(3,SCREEN_WIDTH)
img.set(4,SCREEN_HIGHT)
CENTER_X = SCREEN_WIDTH/2
CENTER_Y = SCREEN_HIGHT/2
BALL_SIZE_MIN = SCREEN_HIGHT/10
BALL_SIZE_MAX = SCREEN_HIGHT/3

# Filter setting, DONOT CHANGE
hmnL = 25.5
hmxL = 53.5
smnL = 134.6
smxL = 225.67
vmnL = 120.36
vmxL = 220.3

# camera follow mode:
# 0 = step by step(slow, stable),
# 1 = calculate the step(fast, unstable)
follow_mode = 1

CAMERA_STEP = 2
CAMERA_X_ANGLE = 20
```

```
CAMERA_Y_ANGLE = 20


MIDDLE_TOLERANT = 5

PAN_ANGLE_MAX   = 170

PAN_ANGLE_MIN   = 10

TILT_ANGLE_MAX  = 150

TILT_ANGLE_MIN  = 70

FW_ANGLE_MAX    = 90+30

FW_ANGLE_MIN    = 90-30


SCAN_POS = [[20, TILT_ANGLE_MIN], [50, TILT_ANGLE_MIN], [90, TILT_ANGLE_MIN],
[130, TILT_ANGLE_MIN], [160, TILT_ANGLE_MIN],
        [160, 80], [130, 80], [90, 80], [50, 80], [20, 80]]


bw = back_wheels.Back_Wheels()

fw = front_wheels.Front_Wheels()

pan_servo = Servo.Servo(1)

tilt_servo = Servo.Servo(2)

picar.setup()


fw.offset = 0

pan_servo.offset = 10

tilt_servo.offset = 0


bw.speed = 0

fw.turn(90)

pan_servo.write(90)

tilt_servo.write(90)
```

```
motor_speed = 60


def nothing(x):
    pass


def main():
    pan_angle = 90            # initial angle for pan
    tilt_angle = 90           # initial angle for tilt
    fw_angle = 90

    scan_count = 0
    print("Begin!")
    while True:
        x = 0           # x initial in the middle
        y = 0           # y initial in the middle
        r = 0           # ball radius initial to 0(no balls if r < ball_size)

        for _ in range(10):
            (tmp_x, tmp_y), tmp_r = find_blob()
            if tmp_r > BALL_SIZE_MIN:
                x = tmp_x
                y = tmp_y
                r = tmp_r
                break

        print(x, y, r)

        # scan:
        if r < BALL_SIZE_MIN:
```

```
            bw.stop()
            if scan_enable:
                #bw.stop()
                pan_angle = SCAN_POS[scan_count][0]
                tilt_angle = SCAN_POS[scan_count][1]
                if pan_tilt_enable:
                    pan_servo.write(pan_angle)
                    tilt_servo.write(tilt_angle)
                scan_count += 1
                if scan_count >= len(SCAN_POS):
                    scan_count = 0
            else:
                sleep(0.1)


        elif r < BALL_SIZE_MAX:
            if follow_mode == 0:
                if abs(x - CENTER_X) > MIDDLE_TOLERANT:
                    if x < CENTER_X:                    # Ball is on left
                                                         print('Ball is on the left')
                    else:                               # Ball is on right
                        print('Ball is on the right')


                if abs(y - CENTER_Y) > MIDDLE_TOLERANT:
                    if y < CENTER_Y :                   # Ball is on top
                        tilt_angle += CAMERA_STEP
                        #print("Top    " )
                        if tilt_angle > TILT_ANGLE_MAX:
                            tilt_angle = TILT_ANGLE_MAX
                    else:                               # Ball is on bottom
```

```
                tilt_angle -= CAMERA_STEP
                #print("Bottom ")
                if tilt_angle < TILT_ANGLE_MIN:
                    tilt_angle = TILT_ANGLE_MIN
            else:
                delta_x = CENTER_X - x
                delta_y = CENTER_Y - y
                #print("x = %s, delta_x = %s" % (x, delta_x))
                #print("y = %s, delta_y = %s" % (y, delta_y))
                delta_pan = int(float(CAMERA_X_ANGLE) / SCREEN_WIDTH * delta_x)
                #print("delta_pan = %s" % delta_pan)
                pan_angle += delta_pan
                delta_tilt = int(float(CAMERA_Y_ANGLE) / SCREEN_HIGHT * delta_y)
                #print("delta_tilt = %s" % delta_tilt)
                tilt_angle += delta_tilt

                if pan_angle > PAN_ANGLE_MAX:
                    pan_angle = PAN_ANGLE_MAX
                elif pan_angle < PAN_ANGLE_MIN:
                    pan_angle = PAN_ANGLE_MIN
                if tilt_angle > TILT_ANGLE_MAX:
                    tilt_angle = TILT_ANGLE_MAX
                elif tilt_angle < TILT_ANGLE_MIN:
                    tilt_angle = TILT_ANGLE_MIN

            if pan_tilt_enable:
                pan_servo.write(pan_angle)
                tilt_servo.write(tilt_angle)
            sleep(0.01)
```

```
        fw_angle = 180 - pan_angle
        if fw_angle < FW_ANGLE_MIN or fw_angle > FW_ANGLE_MAX:
            fw_angle = ((180 - fw_angle) - 90)/2 + 90
            if front_wheels_enable:
                fw.turn(fw_angle)
            if rear_wheels_enable:
                bw.speed = motor_speed
                bw.backward()
        else:
            if front_wheels_enable:
                fw.turn(fw_angle)
            if rear_wheels_enable:
                bw.speed = motor_speed
                bw.forward()
    else:
        bw.stop()


def destroy():
    bw.stop()
    img.release()


def test():
    fw.turn(90)


def find_blob() :
    radius = 0
    # Load input image
    _, bgr_image = img.read()
```

```
    orig_image = bgr_image


    bgr_image = cv2.medianBlur(bgr_image, 3)


    # Convert input image to HSV
    hsv_image = cv2.cvtColor(bgr_image, cv2.COLOR_BGR2HSV)


    thImage = cv2.inRange(hsv_image, (hmnL, smnL, vmnL), (hmxL, smxL, vmxL))


    gausF = cv2.GaussianBlur(thImage, (9, 9), 2, 2)


    circles = cv2.HoughCircles(thImage, cv2.HOUGH_GRADIENT, 1, 120, 100, 20, 10, 0)
    circles = np.uint16(np.around(circles))
    # Loop over all detected circles and outline them on the original image
    all_r = np.array([])
    # print("circles: %s"%circles)


    if circles is not None:
        try:
            for i in circles[0, :]:
                # print("i: %s"%i)
                all_r = np.append(all_r, int(round(i[2])))
            closest_ball = all_r.argmax()
            center = (int(round(circles[0][closest_ball][0])),
int(round(circles[0][closest_ball][1])))
            radius = int(round(circles[0][closest_ball][2]))
            if draw_circle_enable:
                cv2.circle(orig_image, center, radius, (0, 0, 255), 2)
        except IndexError:
```

```
            pass
            # print("circles: %s"%circles)


    # Show images
    if show_image_enable:
    cv2.namedWindow("Gaussian Filter for threshold image", cv2.WINDOW_NORMAL)
    cv2.imshow("Gaussian Filter for threshold image", thImage)
    cv2.namedWindow("Detected green circles on the input image",
cv2.WINDOW_NORMAL)
    cv2.imshow("Detected green circles on the input image", orig_image)


    k = cv2.waitKey(5) & 0xFF
    if k == 27:
        return (0, 0), 0
    if radius > 3:
        return center, radius
    else:
        return (0, 0), 0



if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```