The Luna T-Ball Robot

Tennis Ball Recognizing and fetching Robot

Image Processing course Final Project

1. Introduction

This document specifies the "Luna Tennis Ball Recognizing Robot" project specification, in accordance to the Image Processing 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.

2. Project tools and hardware

2.1. **Hardware**

- Raspberry Pi 4 8GB – 60$
- USB web camera – 6$

Total amount(approximately) - 66$

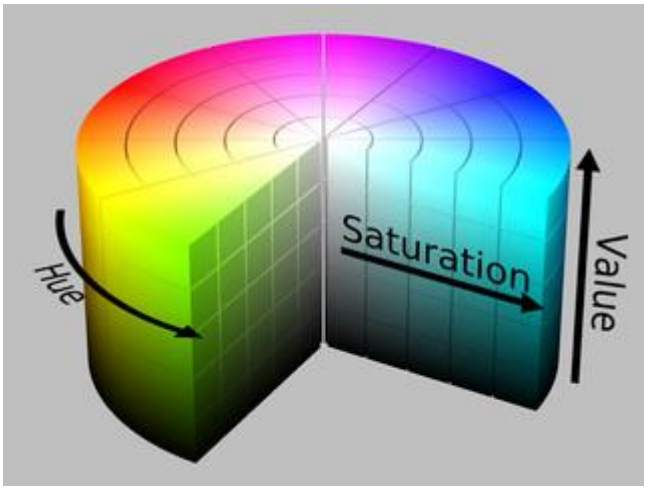
2.2. **Software**

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

3. HSV Masking and Blob finding

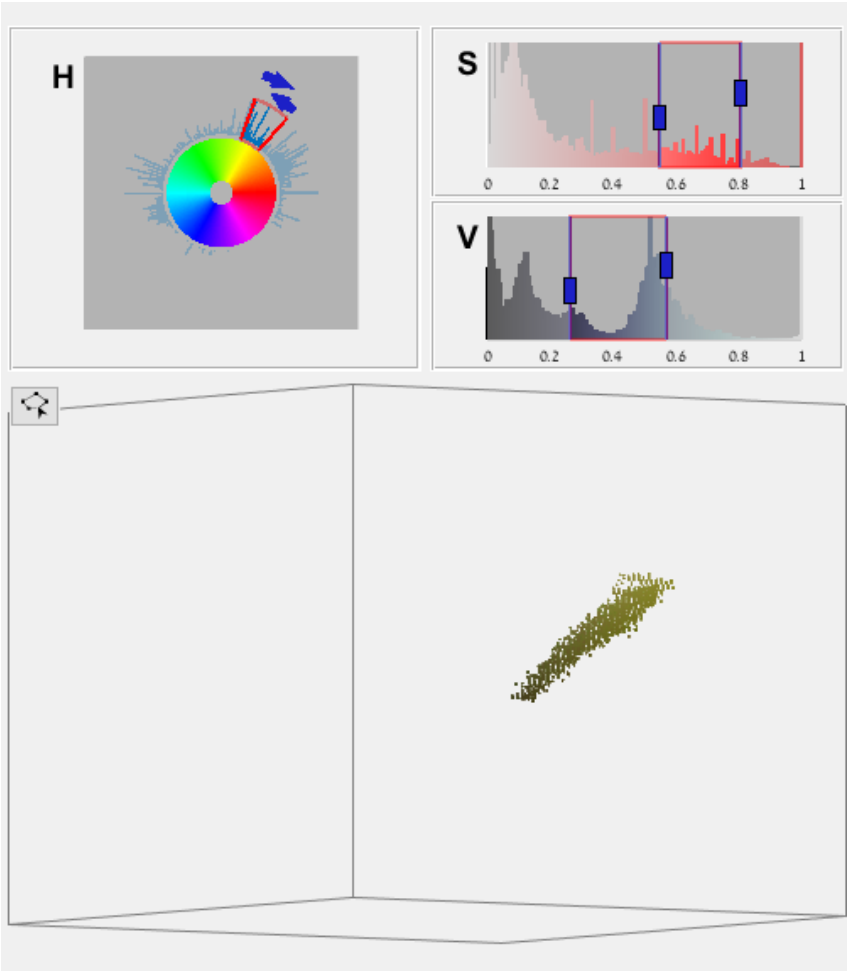
3.1. HSV color space

HSV (hue, saturation, value) color space is a model to represent the color space similar to the RGB color model. Since the hue channel models the color type, it is very useful in image processing tasks that need to segment objects based on its color. Variation of the saturation goes from unsaturated to represent shades of gray and fully saturated (no white component). Value channel describes the brightness or the intensity of the color. Next image shows the HSV cylinder.



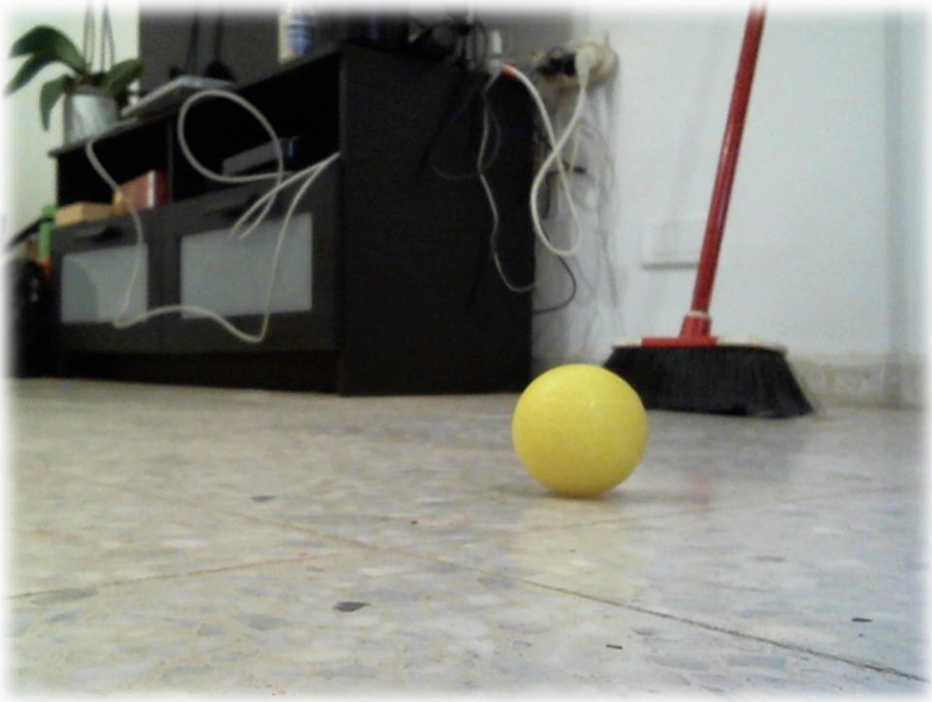
3.2. In order to recognize the yellow/green color, I used the HSV color space and extracted the lower and upper limits of those colors thresholds.

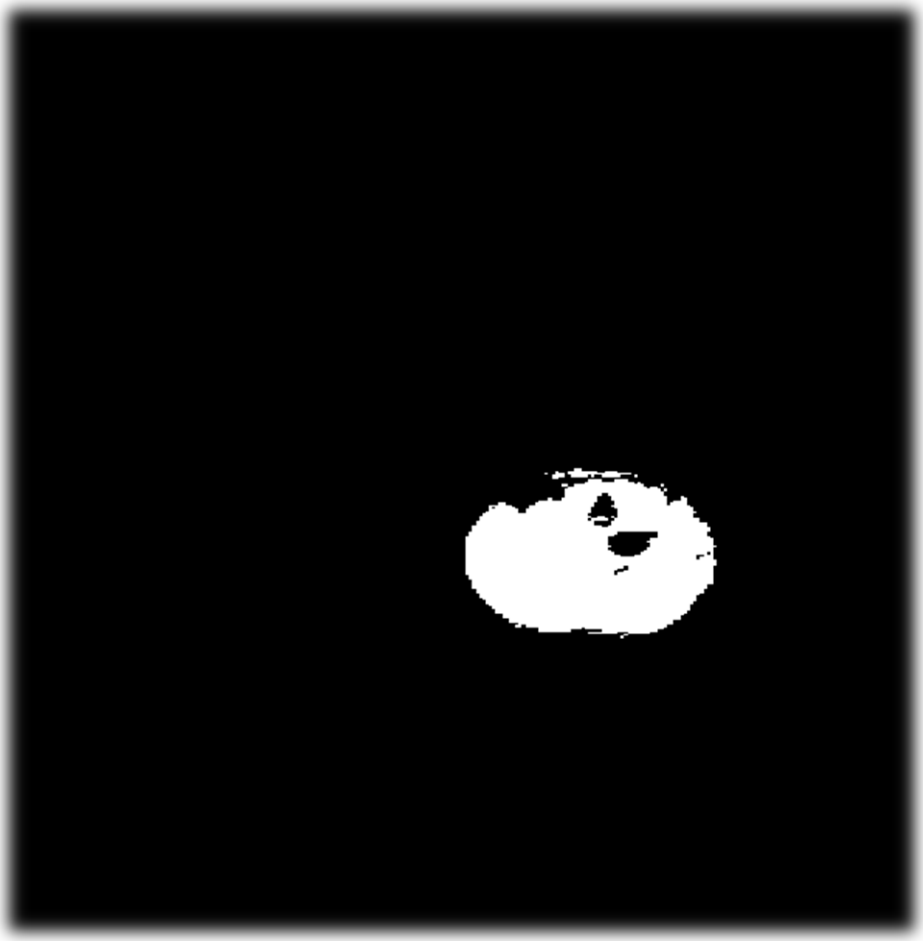
3.3. The values where extracted manually and implemented in the code using Matlab threshold tool.



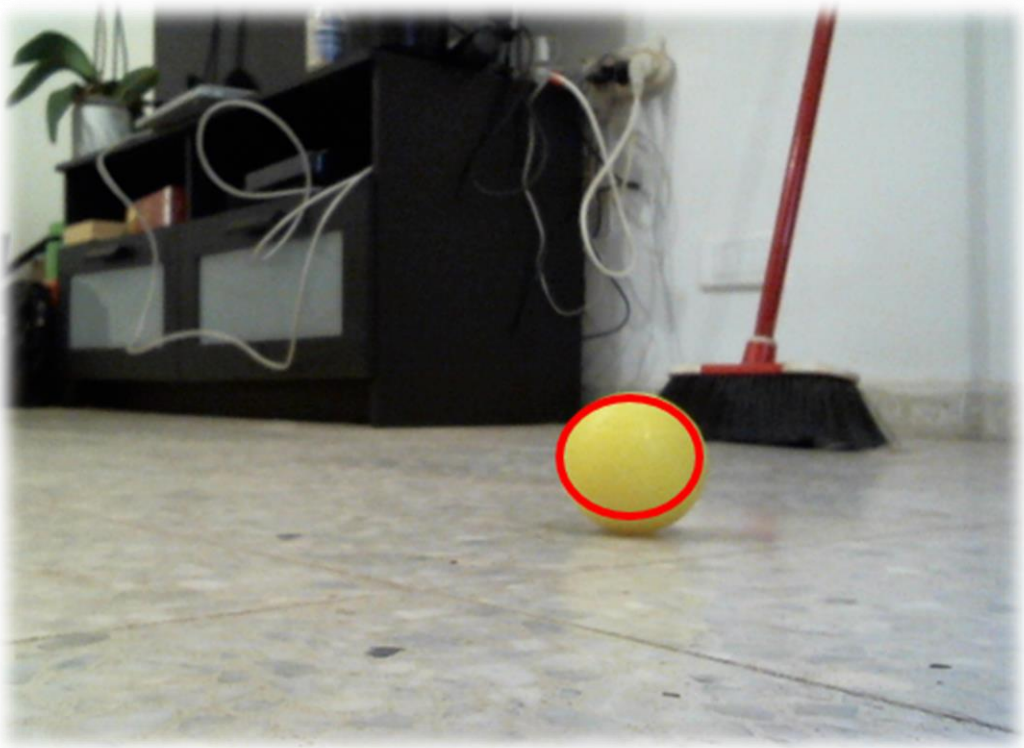
3.4. After applying the mask and extracting the HSV image, I am applying a Gaussian filter on the image to smooth it and get read of small elements and showing it as a binary image of the chosen threshold.

- Gaussian noise reduction 9x9 mask
  Hue levels (25.5, 53.5)
  Saturation levels (134.6,255.67)
  Value levels (120.36, 220.3)





3.5. After getting smooth black and white filtered image, I can extract the blob from it and by applying python function `cv2.HoughCircles()` I can find circels in the picture and calculate their values (radius, center).



4. Course material that are implemented in this project

4.1. Image processing masking and filtering

- We learned about masking for better recognition of objects in images, the importance of using more than one mask in order to get to a better result.
- How noise affect the quality of the image and how to deal with it.

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. Raspberry Pi tutorial

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

6. The Code (adjusted for only image processing) – Python 3.8

```
import time
import cv2
import numpy as np


#show_image_enable    = True
draw_circle_enable  = True
camera_port = 1
kernel = np.ones((5,5),np.uint8)

img = cv2.VideoCapture(camera_port, cv2.CAP_DSHOW)


SCREEN_WIDTH = img.get(3)
SCREEN_HIGHT = img.get(4)
CENTER_X = SCREEN_WIDTH/2
CENTER_Y = SCREEN_HIGHT/2
BALL_SIZE_MIN = SCREEN_HIGHT/10
BALL_SIZE_MAX = SCREEN_HIGHT/3
MIDDLE_TOLERANT = 5

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


while (img.isOpened()):
    ret, 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)

    # Threshold the HSV image, keep only the green pixels
    thImage = cv2.inRange(hsv_image, (hmnL, smnL, vmnL), (hmxL, smxL,
vmxL))
    # Combine the above two images

    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([])


    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)
                time.sleep(0.1)
        except IndexError:
            print('No ball was found, I will keep looking')
            continue
        else:
            pass

    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 = center , radius
        if tmp_r > BALL_SIZE_MIN:
            x = tmp_x
            y = tmp_y
            r = tmp_r
            break
    print(x, y, r)

    if r < BALL_SIZE_MIN:
        print('The ball is far away')
    elif r < BALL_SIZE_MAX:
        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:
                print('Ball is on top')
            else:
                print('Ball is in the bottom')

    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)

    print('press "q" to exit the program')
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

img.release()
cv2.destroyAllWindows()
```