EE 144 - Lab 3: Color Detection and Tracking

April 26

1 Introduction

In this lab, the goal is to become familiar with color-based tracking, and with controlling the robot to move according to visual input.

1.1 Pre-lab assignments

Read on Hue Saturation Value (HSV) and Red Green Blue (RGB) color spaces, if you are not already familiar with them.

1.2 Lab assignments

The following are the assignments for the lab. You should ideally be able to complete the assignments in the 3-hour period of the lab session, but you may also continue your work during the following days if needed. You should report your work in an **individual** lab report, which is due one week after the start of your lab session. In your submission you should also include your **commented** code, as described in the instructions.

Assignment 1: Color detection in the HSV color space

You are given the file `HSV_color_detection.m` (as in the previous labs, you should change the IP address in the initialization, to make this work on your system). To run this script, first start the virtual machine, and run the Empty Gazebo simulator. The code will create a few colored balls in the environment, and will display the images that the robot's camera is recording in this environment.

Your goal is to write code that will detect the green ball in the images using thresholding in the HSV color space, and make the robot turn in place so that the green ball appears in the middle of the image.

One way to achieve this is by using the following steps for each image:

- use thresholds on the Hue of each pixel, to create a black-and-white (also called binary) image, where all pixels deemed to belong to the green ball are set to one, and all other pixels to zero.
- use the function `regionprops` to find the centroid of the image region that is identified as belonging to the green ball.
- define the rotational velocity of the robot using a "P-controller" whose input is the difference between the centroid column and the middle column of the image.

Note: when processing images in Matlab, it is important to use vectorization when possible, to speed up computation. For example, let's say that we are given a matrix `A`, and we want to generate a matrix `B`, where each element of `B` equals one if the corresponding element of `A` is in the interval (`t1, t2`), and zero otherwise. This can be implemented, using for-loops, as follows:

```
B = zeros(size(A));
for i = 1:size(A,1)
    for j = 1:size(A,2)
        if A(i,j)>t1 && A(i,j)<t2
```

```
            B(i,j)=1;
        else
            B(i,j)=0;
        end
    end
end
```

However, we can alternatively implement the same thing, as follows:

```
B = (A>t1).*(A<t2);
```

Here, the expression `(A>t1)` generates a binary matrix, whose elements equal one if the corresponding element in `A` is larger than t1, and zero otherwise. Similarly, `(A<t2)` generates a binary matrix whose elements equal one if the corresponding element in `A` is smaller than t2, and zero otherwise. The element-wise product of these two matrices (obtained via the Matlab operator `.*`) is the desired matrix `B`. This second version of the code is *significantly* faster in Matlab, because it allows the use of Matlab's built-in vectorized/optimized code. In general, whenever we can replace for loops with vectorized expressions, we can achieve significant speedup.

Since in this lab we are interested in controlling the motion of a robot using image input, we should make sure that our code is computationally efficient. If significant latency exists in the control loop, the system may become unstable.

Assignment 2: Robot tracking

You are given the Matlab script `robot_following.m`. To run this script, first start the virtual machine, and run the Empty Gazebo simulator. In this code, we are generating a second robot in the environment, and our goal is to have the Turtlebot track it as it moves in space.

The code initializes the simulation environment, and uses the function `find_robot`, to detect the pixels in an image that belong to the robot. This function is tailor-made for the specific characteristics of the Empty-World simulator. It assumes that the color of the background is perfectly known, and any pixel that does not have that color is assumed to belong to the robot. Once these pixels are identified, the function returns the column coordinate of the robot's centroid, and the area of the image region that belongs to the robot.

Your goal is to write code that will make the Turtlebot track the robot as it moves in space. We would like the robot to (i) appear as close to the image center as possible, and (ii) have an area in the image as close as possible to 4000 pixels. These objectives can again be accomplished by using Proportional controllers for the rotational velocity (to achieve objective (i)), and for the linear velocity (to achieve objective (ii)).

For this assignment, the code should also be able to handle the case where the robot does not appear in the image (e.g., the robot moved too fast and we lost it). One way to address this is to have the robot rotate in place, with a constant rotational velocity, until it re-detects the robot.