CS/ME/ECE/AE/BME 7785

Lab 1

Due: January 28, 2022 at 4 pm

Overview

The objective of this lab is to get you familiar with the robot and to get started writing code. Specifically, this lab consists of two parts:

- **Part 1 (individual)**: run provided Turtlebot3 code to get familiar with running the robot
- **Part 2 (group)**: create your own image processing script that enables the robot to identify predetermined patterns in its camera image.

For Part 2 we have provided two variants of the assignment (Option A and Option B) that use slightly different capabilities of OpenCV. Option A uses an offboard dataset, while Option B is on the robot. Either option is valid and you only need to complete one. Both will be useful, as you will reuse this code in Lab 2 to have your robot move to track whichever object you have chosen. The code for Option A is more computationally expensive than Option B.

Note that regardless of which option you choose for Part 2, Part 1 will take far less time than Part 2, so budget your time wisely.

Part 1 is an individual assignment, each person must demonstrate ability to run the robot from a machine that they have access to. You can use your own laptop or the lab machines. You can not use your partner's laptop to demo.

We strongly encourage you to use all available resources to complete this assignment. This includes looking at the sample code provided with the robot, borrowing pieces of code from online tutorials, and talking to classmates. You may discuss solutions and problem solve with other groups in the class, but each group must submit their own solution. Multiple groups should not jointly write the same program and each submit a copy, this will not be considered a valid submission.

Submission instruction and grading details are included at the end of the document.

Part 1: Interfacing with the Turtlebot3

All instructions for the Turtlebot3 are available online at

http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

Complete the following steps:

1. The turtlebot3s in the lab are set up to connect to GTother. For security reasons they must be authenticated by a user with a Georgia Tech account to access the internet. To do this...
   a Use the LCD screen on the Turtlebot to find the IP address and MAC address for that robot.
   b Log onto auth.lawn.gatech.edu
   c Enter the appropriate information to give the robot access to the internet.
2. Make sure you can access the Raspberry Pi on the robot. You will need to SSH (secure shell) into the robot using the command,

   `ssh burger@192.168.1.152`

   (The IP 192.168.1.152 should be replaced by your Raspberry Pi's IP or hostname)

   The password is "**burger**" for all the robots. Once entering the password you should see the terminal environment on the Raspberry Pi of your Turtlebot3!
3. Complete the Quick Start guide on the above website (listed as item 3 on the left side menu) and verify that you are able to teleoperate the robot using the keyboard. Optionally, feel free to run any other examples, such as Turtlebot Follower to test out the robot functionality.

Note: On the robot itself, edit the `~/.bashrc` file such that the IP address listed under `ROS_MASTER_URI` is the IP of the computer where you are running `roscore`.

Note 2: Don't forget to source `~/.bashrc` whenever you edit that file to make the changes take effect.

Part 2 Option A: Object Following (offboard template identification)

Objective: Use the `matchTemplate()` capability within OpenCV to locate diamonds in a dataset of images we collected from the robot.

In this variant of the second part of the lab, you will only use OpenCV applied to a dataset of images collected by us from the robot camera. Please use python3 for this option. The robot is not needed for this part of the lab. Feel free to use any additional python libraries you want, such as numpy, etc, if you do, make sure to include a requirements.txt file along with your submission.

Write a python script called `diamond_finder.py` to detect red diamond shapes in images using the `cv.matchTemplate()` functionality in OpenCV.

diamond_finder.py This script should load images from a directory called `input_imgs`, process those images, and then output annotated images to a directory called `output_imgs`.

To do this, create the script file named `diamond_finder.py`. To run it, you'll want to make the file an *executable*. To do this you must first make sure you have the type of environment being used in the first line of your code. For python this typically is,

```
#!/usr/bin/env python
```

Then to make the file executable, using the command line in the directory (or with the full path specified) where your file is stored type,

```
chmod +x diamond_finder.py
```

You may now run your python script from the command line, when in the directory where it is stored, by giving the command,

```
./diamond_finder.py
```

There are many online resources that show how to read, write and process image files using OpenCV. You should search around by yourself, but one example is:

https://docs.opencv.org/4.5.0/db/deb/tutorial_display_image.html

For each file you read in, use the template image provided with this lab to locate diamond shapes in the image using OpenCV's template matching functionality. Again, there are multiple tutorials available, one is:

https://docs.opencv.org/4.5.2/d4/dc6/tutorial_py_template_matching.html

For each image processed by your algorithm:

- if no diamond was found, output the image to the `output_imgs` as-is, without modification
- if diamonds were found, then annotate the image by drawing a bounding box around each located region and save the modified image to `output_imgs`. Additionally print the coordinates of each bounding box to the terminal in the format `[image_name bounding_box_x bounding_box_y]`.

Once all images are processed, your script should return a dictionary of lists of lists, where the key is the name of the image, and the value is a list of lists of points associated with that image. e.g. if image6.jpg has three diamonds in it located at pt1, pt2, and pt3, the return should be:

```
dictionary = {'image6.jpg':[[pt1_x, pt1_y], [pt2_x, pt2_y], [pt3_x, pt3_y]]}
```

The template matching tutorial above only works when the template image and the source image are scaled similarly. This will cause the template matching algorithm to miss many diamonds in your images because they might be at different scales. Modify your code for scale invariant template matching by scaling the source image. You can find examples online, to encourage active googling we will not provide a link here!

A note on performance: 100% accuracy with this approach is not expected, nor is it necessary. In the real world robots never have 100% reliable inputs to work with, and we must structure our behavior code to account for this. As a result, in this lab, we are not trying to achieve 100% performance (though if you want to, it's definitely possible). If your performance is particularly low, you can prefilter your images by testing out blurring, sharpening, improving contrast, running edge detection prior to template matching, etc. All of these things are relatively easy to try with OpenCV and involve just a few lines of code. Feel free to explore as much as you want, there are many possible ways to achieve good performance on this task.

Part 2 Option B: Object Following (ball finding on the robot)

Objective: Use the `HoughCircles()` capability within OpenCV to locate a round ball real-time images streaming from the robot.

In this variant of the second part of the lab, you will use OpenCV to locate a ball in the robot's camera frame. This is a valuable step towards Lab 2, in which you will add driving capabilities to your code. Feel free to use any additional python libraries you want, at the very least cv2 (OpenCV) and numpy, please include a requirements.txt file with your submission if you use any libraries outside of these.

Create a new python script called **find_ball.py**. This script should receive images from the webcam on your laptop and track the location of a specific ball in the frame. Once you have made the script it is often useful to make it an *executable*. To do this you must first make sure you have the type of environment being used in the first line of your code. For python this typically is,

```
#!/usr/bin/env python
```

Then to make the file executable, using the command line in the directory (or with the full path specified) where your file is stored type,

```
chmod +x ball_follower.py
```

You may now run your python script from the command line, when in the directory where it is stored, by giving the command,

```
./ball_follower.py
```

Some example code to capture and display your webcam with OpenCV in python can be found at

https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_gui/py_video_display/py_video_display.html

After grabbing the image, process the image to look for circles. An easy place to start is to use the HoughCircles() functionality within OpenCV:

```
circles = cv2.HoughCircles(cv_image, cv2.HOUGH_GRADIENT, 1, 90, param1=70, param2=60,
                           minRadius=50, maxRadius=0)
```

As described in class, this alone will likely not be enough. Be sure to prefilter your images (blurring, sharpening, improving contrast, thresholding color, etc). You are not required to use Hough circles, if you prefer to try a different method that is completely acceptable so long as the ball is tracked.

Once you have located the ball in an image, print the pixel coordinate of the ball and display some sort of marker or bounding box on the image itself.

We will provide balls of several sizes and colors for your use in the lab. Feel free to use any of them, or a different ball of your choosing. For this lab, the ball can be at any distance from the webcam that you choose (not taking up the entire frame).

Grading Rubric

The lab is graded out of 100 points, with the following point distribution:

Successfully run teleop or example code on the robot (individual assignment)	25%
Find >65% of your chosen shapes in images	65%
Print the pixel location of each identified shape	5%
Annotate output images with bounding boxes or markers	5%

Submission

Part 1: Perform a live demonstration of you running the robot to one of the course staff by the listed deadline. You can use your own laptop or the lab machines, you can not use your partner's laptop to demo Part 1. There is no code to submit.

Part 2:

1. Perform a live demonstration of you running the robot to one of the course staff by the listed deadline.
2. Put the names of both lab partners into the header of the python script. Put your python script and any supplimentary files, in a single zip file called Lab1_LastName1_LastName2.zip and upload on Canvas under Assignments–Lab 1.
3. Only one of the partners needs to upload code.

We will set aside class time and office hours on the due date for these demos, but if your code is ready ahead of time we encourage you to demo earlier in any of the office hours (you can then skip class on the due date). Office hour times are listed on the Canvas homepage.