CSCE 439/839: Robotics
Lab 3: Ball Detection

Instructor: Carrick Detweiler
carrick _at_ cse.unl.edu
University of Nebraska-Lincoln
Fall 2022

Started: Friday, March 25, 2022
Lab Checkpoint (Up through Sec. 5): Friday, April 1, 2022
Lab Due Date: Friday, April 8, 2022

1 Overview

In this lab, you will connect and configure a Raspberry Pi to work with your robot. You will then use code to visually localize a ball and perform visual servoing to follow the ball at a fixed distance. I'm giving you the ball detection code, but you will need to adapt, characterize, and integrate it into your system.

Make sure to read through the entire lab before starting. This lab has checkpoints. In lab on the date of the checkpoint you will be responsible for showing the instructor your progress. You do not have to have the section written up yet (although that might not be a bad idea), but you need to be able to demonstrate all the tasks in the section.

Before starting you should read through the whole lab. Some parts can be done in parallel, while some sections rely on the completion of previous sections. You should discuss your plan of attack for the lab in your group and decide how you will work together and divide the work. Everyone, however, is responsible for knowing about all sections of the lab. In addition to completing the lab report, on the due date, you will demonstrate what you accomplished to the instructor.

Also, the points for this lab do not add up to 100, but each lab is worth the same amount as any other lab even if the points are not equal.

2 Raspberry PI Setup (5 pts.)

Your group has a Raspberry Pi. In this section, you should follow the instructions given in lab to mount the Pi, camera, and battery to your robot.

Question: Document and describe the setup process and any challenges you encountered. (Pictures are very useful!)

You will need to retune your Balboa to be able to balance with the much larger amount of weight.

Question: Describe how you tuned the Balboa to be able to balance. Discuss how well it balances and drives around as compared to the configuration without all the extra stuff.

Some important reminders regarding your Raspberry Pi:

- You should be able to use ssh or a vncviewer (using port 5900) to log into your Raspberry Pi at the hostname of your Pi when you are on the UNL network. Note that this is logging in natively to the Pi, not into the Docker container.
- If you are on an off campus network, you will need to connect the Pi to a monitor, keyboard, and mouse configure it to connect to your own WiFi. This might then break how it connects to the UNL network (or maybe not). So be prepared for some extra debugging time if you plan to do this. As described below, using bag files to do some of the work without the Pi is a useful technique.
- Once you launch the docker container (in the next section), you can then use vncviewer (using port 5901) to connect to the Docker container. You can do this from your own computer.

- Make sure to issue the `sudo shutdown now` command before disconnecting power from the Pi. If you do not, you may corrupt the file system or you may not be able to log in properly.
- You should set the password on your Pi and also the Docker container to something your group knows and not just the default password as this could cause confusion among groups.

3 Software Setup (5 pts.)

We are going to run Docker on the Raspberry Pi, which will allow you to do development on the Raspberry Pi using the same configuration you have been using on your own computers. That way you can continue to test and develop on your computer and then transfer it to the Pi for testing (e.g. using scp or rsync).

To use cameras in ROS, we need to install the package ros-melodic-usb-cam. Modify your Dockerfile so that it also apt-get's this package. Copy the Docker and other files over to the Raspberry Pi. You will then need to build the Docker image on the Raspberry Pi and also rebuild it on your own computers.

Once this is complete, download the code for this lab (called `ballDetector`). Put this in the shared `catkin_ws/src` directory.

To be able to access the USB camera in Docker on the Raspberry Pi we need to pass the `--privileged` flag as follows when starting the container:

```
sudo docker run --rm  --privileged -v $PWD/shared:/home/shared -p 5901:5901 -it ros
```

Note that this will allow access to the webcam on the Raspberry Pi, however, this will not work on Windows or MacOS machines[1].

Once you have your Docker container running, build the ball detector code with `catkin_make`. Note that this must be done within the docker container (use your VNC viewer to connect to the Docker container).

You can now test the camera by running:

```
roslaunch ball_detector camtest.launch
```

Note that you will likely need to resize the camera view window to see the whole image.

Question: Make sure to discuss these steps and any issues you encountered and how you overcame them.

4 Bag Files and Turtles! (10 pts.)

ROS has a tool called `rosbag` that allows you to record and replay messages. Since you can't easily use the camera on your own computer, it is often useful to record bag files on the Raspberry Pi and then replay and work with them on your own computer, especially when you do not have access to the robot.

Read the tutorial on the rosbag command line tutorial for this part at: `https://wiki.ros.org/rosbag/Commandline`. On the course website there are some sample bag files. Download one of the turtle bag files.

Question: Which bag file did you pick? What topics are contained in the bag file? How did you figure this out?

Question: Start up the basic turtlebot (`rosrun turtlesim turtlesim_node`). Playback the bag. How did you playback the bag? What letter does the turtle draw?

Question: Restart the turtlebot and also startup the teleop (`rosrun turtlesim turtle_teleop_key`). Record just the `/turtle1/pose` topic in a rosbag (and no others) while you draw (you guessed it) "CS." How did you record just the one topic? Include this bagfile in your submission.

Passing the flag `-r` when playing a bag file allows you to adjust the playback speed. This can be particularly useful when doing vision processing (like in the next sections) if you have a slower virtual machine.

Question: How can you play it back at half speed? Give the full command you use.

[1]If you are interested, the appendix contains info on how to use some scripts to connect a camera in Windows or MacOS.

5 Ball Detection (15 pts.)

In the code download for this lab there is code to detect balls (called `ballDetector`). In the `launch` directory, there are a number of launch files. The first, that you have already used `camtest.launch` starts up the camera and a window to display the view from the camera. The second, `ballDetector.launch`, just launches the and ball detector code. The third, `displayDebugImages.launch`, will display three different images you can use for debugging. There is also a launch file called `all.launch` that will launch all of these together and includes the HSV selector. You will likely want to modify or put together your own launch files based on your workflow.

With all this running the Pi can get a bit slow, but closing some of the display windows that you do not need can speed things up. In addition, in the future when you are running lots of other nodes and are done tuning the detector, you can disable the debug images and information by commenting out the `#define BALLDETECTOR_DEBUG` line in the `ballDetector.cpp` file.

The ball detector works by first converting the image to HSV color space. It then performs thresholding to filter out all pixels that are between a low and high HSV threshold. These values can be configured in the launch file (see `ballDetector.launch` for an example) and can be dynamically changed by running the command `rosrun ball_detector configGUI.py`. It then searches for the largest group of connected pixels that have similar height and width (as a crude approximation to find circular groups of pixels). The largest group that meets this criteria is selected as the "ball" in the image and a message is sent out with this ball location.

The debug images display the HSV image, thresholded image (white pixels indicate those that are between the low and high thresholds), and a marked up image with white pixels indicating the boundaries of connected groups. In addition, in this image the final ball location is marked with a circle. Use these images and the configGUI to pick good HSV low and high thresholds.

Hint: If you record some bag files with different lighting and different color balls it becomes much easier to loop these and work on it without needing to have the Raspberry Pi and the balls.

Question: How did you go about picking the HSV thresholds for various balls? Do this for a couple of different color balls and report on the results. How does lighting affect the values?

Question: How many locations per second does the ball detector detect? Does it change based on different HSV thresholds? Make sure to describe the systems you tested this on (e.g. on the Pi, your laptop from 1990, the latest and greatest super computer, etc).

Question: Do experiments to characterize and calibrate range to the ball based on the reported radius. Derive an equation that fits your data so that you can get a distance estimate to the ball when you detect it. Characterize the performance of the range estimation.

Question: **839 majority groups only:** This does not need to be completed by the checkoff. Write a new node that subscribes to the HSV debug image and automatically tunes the HSV values when a ball is placed on a white background (it is ok to assume that the ball is at least a certain size in the image). Describe the details of this node and how well it works compared to manual tuning.

6 Visual Servo (15pts.)

In the previous lab, you implemented a reactive controller that maintained a fixed distance from an object based on the IR sensor readings. In this lab, you will implement a similar behavior by visually servoing to center and maintaining a fixed distance from a ball. You get to pick whatever color ball you would like to use. The visual servo code should maintain a fixed distance from the ball and also turn to follow it.

Question: Describe your visual servo implementation.

Question: How well can you perform visual servoing? If you start from far away, how quickly can you get to the target distance? Include useful plots to help characterize the performance (e.g. estimated distance over time). The characterization of the servoing behavior should be detailed.

Question: What do you do if you lose sight of the target? How do you determine if you have lost the target?

7 To Hand In

You should designate one person from your group as the point person for this lab (each person needs to do this at least once over the semester). This person is responsible for organizing and handing in the report, but everyone must contribute to writing the text. You should list all group members and indicate who was the point person on this lab. Your lab should be submitted on Canvas. Submit a **pdf** of your report and a zip file of the workspaces you wrote and any other source code used in this lab.

Your lab report should have an introduction and conclusion and address the various questions (highlighted as ***Question:***) throughout the lab in detail. It should be well written and have a logical flow. Including pictures, charts, and graphs may be useful in explaining the results. There is no set page limit, but you should make sure to answer questions in detail and explain how you arrived at your decisions. You are also welcome to add additional insights and material to the lab beyond answering the required questions. The clarity, organization, grammar, and completeness of the report is worth **15 points** of your lab report grade.

In addition to your lab report, you will demonstrate your system and what you accomplished up to this point to the instructor at the beginning of lab on the due date. This is worth **15 points** of your overall lab grade. You do not need to prepare a formal presentation, however, you should plan to discuss and demonstrate what you learned and accomplished in all sections of the lab. This presentation should take around 5 minutes.

Question: You must include your code with the lab report. Note that you will receive deductions if your code is not reasonably well commented. You should comment the code as you write it, do not leave writing comments until the end.

Question: Include an `rqt_graph` of your final system and comment on your overall system architecture.

Question: For everyone in your group how many hours did each person spend on this part and the lab in total? Did you divide the work, if so how? Work on everything together? Give details on the hours each group member worked (e.g. keep a list of how many hours per day each person worked and on what).

Question: Please discuss and highlight any areas of this lab that you found unclear or difficult.

8 Appendix: Raspberry Pi Setup

We have already done these setups on your Raspberry Pi, but here are the things we did to make this image work:

- Put a fresh install of Raspberry Pi OS Full (64-bit, version 11.2). The 64bit may be slightly faster on a Pi4 as compared to the 32bit version.
- Set the password, then run the Raspberry Pi Configuration utility and: enable vnc and ssh; set the hostname; and set headless resolution to 1280x720.
- Register the the Pi on the UNL IoT network and give it a meaningful and unique hostname.
- Label the Pi with the hostname
- Run: `sudo apt-get update; sudo apt-get upgrade` to get the latest and greatest.
- Run: `sudo apt-get install arduino screen emacs tigervnc-viewer nmap`
- Run: `pip3 install serial serial-tool opencv-python`
- Get Docker by running: `curl -fsSL https://get.docker.com -o get-docker.sh`
- `sudo sh get-docker.sh`
- `sudo pip3 install docker-compose`

9 Appendix: Cameras with Docker

On Windows and MacOS, even with the privileged flag, Docker is unable to connect to USB devices like cameras. It is, however, possible to forward images from a camera over the network from your host machine to the Docker image. This adds latency and is fairly CPU intensive. So it might not work on all machines. For instance, running it this way on the Raspberry Pi adds a couple of second delay. But if you are interested, here is how it works:

- Download the camera server code from the course website. This contains three python scripts.
- On your computer, install opencv for python: `pip3 install opencv-python`
- When running the Docker command, add the flags: `-p 8089:8089` This forwards a port from the container to the host to allow the connection.
- From within the docker container run the `camera_receiver.py` code. This will wait for a connection with the camera.
- On your native machine, run `camera_server.py` If you do not pass any arguments it will list out the cameras on your machine. You then need to run it with a single number as an argument to tell it which camera to use. You might need to use some trial and error to find the right one. Note that at least on MacOS, the native camera is even slower than the external USB camera for some reason.
- If everything works, you should see a live view from the camera from within docker.
- To use the camera in ROS, run `camera_receiver_ros.py` instead after you have started the roscore or your launch file. This will then publish the images to a topic.

There are likely better and faster ways to make this work (and this isn't highly tested, so be warned!). If anyone gets something working that runs well on typical machines, let me know!