Institute for Aerospace Studies
UNIVERSITY OF TORONTO

LABORATORY 3

Monocular Feature Detection and Mapping

ROB521 Autonomous Mobile Robotics Winter 2020

By: Steven Waslander, Emmett Wise, Trevor Ablett - 2020

1 Introduction

This is the third laboratory exercise of ROB521—Autonomous Mobile Robotics. The course will encompass a total of four labs and a design project, all of which are to be completed inthe scheduled practicum periods. Each of the four labs will grow in complexity and intended to demonstrate important robotic concepts presented during the lectures. Our robot of choice: *Turtlebot 3 Waffle Pi* running the operating system *ROS*.

You will be required to hand in a brief lab report and will be graded on this lab.

2 Objective

The objective of this lab is to develop a map of visual features while estimating the robot's pose. In particular, you are

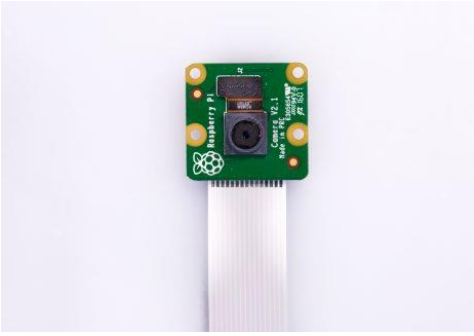
- *To learn about calibrating and using the robot's camera*
- *To extract and track features in the image plane*
- *To estimate the position of landmarks in 3D from monocular image measurements and wheel odometry based robot poses.*

3 Getting Started

3.1 Remote Control of a Robot

You will again use remote control of the robot for motion through the environment. Refer to Lab's 1 and 2 for the details if necessary.

3.2 Accessing Camera Data



The Raspberry Pi Camera Module v2 replaced the original Camera Module in April 2016. The v2 Camera Module has a Sony IMX219 8-megapixel sensor (compared to the 5-megapixel OmniVision OV5647 sensor of the original camera). The Camera Module can be used to take

high-definition video, as well as stills photographs. It's easy to use for beginners, but has plenty to offer advanced users if you're looking to expand your knowledge. You can also use the libraries we bundle with the camera to create effects. Detailed specifications are available : http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_raspi_cam/

To load full robot including lidar, and all other sensors except the camera, run the following command on the robot (remember to ssh in first):

```
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

To just load the camera, again on the robot:

```
$ roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch
```

To view the image data, open rviz and add the image visualization (under "By Topic"):

```
$ rviz rviz
```

To dynamically change camera settings such as exposure, try:

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

3.3 Working with OpenCV

The OpenCV library is a well established package with excellent integration in ROS. It is extremely convenient for experimenting with a wide range of computer vision methods, and is both well documented and accessible in Python for ease of use. We'll be using the Feature2D library in OpenCV, whose documentation can be found here: https://docs.opencv.org/master/da/d9b/group__features2d.html.

There are many interesting tutorials that cover all kinds of applications of OpenCV to computer vision problems.
https://docs.opencv.org/master/d9/df8/tutorial_root.html

Specific tutorials that are useful to try when first learning OpenCV
Changing Contrast and Brightness:
https://docs.opencv.org/master/d3/dc1/tutorial_basic_linear_transform.html

Canny edge detector:
https://docs.opencv.org/master/da/d5c/tutorial_canny_detector.html

We will work with feature extraction, description and matching methods, for which there are a plethora of options. For some interesting tutorials on using these methods, see here:
https://docs.opencv.org/master/d9/d97/tutorial_table_of_content_features2d.html

4 Assignment

4.1 Task 1: Calibrate the camera

Once again, we start the lab with a calibration exercise. This time, you will perform a basic camera calibration using the checkerboard target and predefined calibration routines available with the Turtlebot3. Although all the heavy lifting is done for you, you will need to carefully work through the calibration procedure and present your calibration values and answer some questions in the final report for this lab.

The raspicam_node package contains a calibration file for the raspberry PI camera versions 1 and 2. However, we will generate our own for our camera using the camera_calibration package from ROS.

1) If you haven't done so already, load the camera by running the following command on the robot:

```
$ roslaunch turtlebot3_bringup turtlebot3_rpicamera.launch
```

2) Check that the camera image message is publishing and is visible on the desktop, using rostopic list.

3) Launch the monocular camera calibration tool on the desktop using the following command:

```
$ rosrun camera_calibration cameracalibrator.py --size 8x6 --square 0.0255 image:=/raspicam_node/image camera:=/raspicam_node --no-service-check
```

The target is actually 9x7 checkerboards, but as you can see in the figure below, the ROS camera_calibration package only uses the inner 8x6 grid. The parameter 0.0255 is the length of the checkerboard square for your calibration targets, which was measured to be 25.5 mm. Do check this value if you have a ruler with you. The image and camera arguments must match what is visible in your ros topics.

4) To perform the calibration, move the checkerboard throughout the visible region. The GUI that loads (eventually) has bars for x, y, size and skew variation and you must excite the full range of each of these modes to get a good calibration. A common procedure is to start in the center of the image parallel to the image plane and move forward until the target is detected and takes about half the image. Then move around the edges of the visible region to cover x and y variation. Next move forward and back to the limits of the checkerboard detection range to complete the size variation. Finally move left to right and up and down with different extreme skews, and be sure to tilt the target vertically and horizontally. When all the bars have turned green, the calibrate circle with turn a dark green and you are ready to calibrate.

5) Press calibrate (nothing happens for while), and then press save and commit. Save will store the calibration to /tmp/calibrationdata.tar.gz. After the calibration is complete you will see the calibration results in the terminal and the calibrated image in the calibration window. A successful calibration will result in real-world straight edges appearing straight in the corrected image. A failed calibration usually results in blank or unrecognizable images, or images that do not preserve straight edges.

6) After a successful calibration, you can use the slider at the top of the calibration window to change the size of the rectified image. A scale of

0.0 means that the image is sized so that all pixels in the rectified image are valid. The rectified image has no border, but some pixels from the original image are discarded. A scale of 1.0 means that all pixels in the original image are visible, but the rectified image has black borders where there are no input pixels in the original image.

Example terminal output.

```
D =  [-0.33758562758914146, 0.11161239414304096, -0.00021819272592442094, -3.0291954463305
18e-05]

K =  [430.21554970319971, 0.0, 306.6913434743704, 0.0, 430.53169252696676, 227.22480030078
816, 0.0, 0.0, 1.0]

R =  [1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0]

P =  [1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0]

# oST version 5.0 parameters


[image]

width

640


height

480


[narrow_stereo/left]


camera matrix

430.215550 0.000000 306.691343

0.000000 430.531693 227.224800

0.000000 0.000000 1.000000


distortion

-0.337586 0.111612 -0.000218 -0.000030 0.0000


rectification

1.000000 0.000000 0.000000

0.000000 1.000000 0.000000

0.000000 0.000000 1.000000
```

```
projection
1.000000 0.000000 0.000000 0.000000
0.000000 1.000000 0.000000 0.000000
0.0      000 1.000000 0.000000
```

If you are satisfied with the calibration, click COMMIT to send the calibration parameters to the camera for permanent storage. The GUI exits and you should see "writing calibration data to ..." in the console. Commit adds it to ~/.ros/camera_info where it can be loaded automatically during launch.

The intrinsic camera matrix for the raw (distorted) images has the following parameters.

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

The camera matrix projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx, fy) and principal point (cx, cy).

The distortion parameters are as follows, with the size depending on the selected distortion model. For the "plumb_bob" model, the 5 parameters are: (k1, k2, t1, t2, k3).

The full tutorial on Monocular Camera Calibration tutorial has additional details if you get stuck, and walks you through the process of how to calibrate a single camera.

To complete this section, present your intrinsic calibration parameters to the TAs. You should save your calibration matrix and distortion parameter estimates to the terminal using print statements and record them for your report and subsequent use in Part 4.3.

4.2 Task 2: Extract and match features

In this task, you will build a functional front end for monocular visual odometry and SLAM. You will rely on the OpenCV library to extract, describe and match features, perform outlier rejection and experiment with different settings to fine tune your front end. An example for how to do this is provided below and in the sample code, but please experiment with the options available to you in the features2d package of OpenCV.

To get started, we'll follow a simple path through the extract, describe and match paradigm.

1) Using the sample code provided, apply Harris corner extraction. To do so, you will first have to convert the ROS image to OpenCV format, then convert RGB image to grayscale. Next you will call the appropriate feature extraction method. Finally visualize the results using the provided tools and store a sample image with visualize features for your report. You can refer to this OpenCV documentation on how to do that: https://opencv-python-

tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

2) Next, you will compute descriptors using extracted salient features. Note that this is separate from the previous Harris Corner part. In this part, we'll use ORB feature descriptor, but feel free to try other detector/descriptor pairs. You can always refer to OpenCV documentation: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html

3) Next, you will match the features using the brute force matcher. To do so, we need at least two images to match, and will use the current image and the next image in a typical odometry fashion. You can refer to this OpenCV documentation here for brute force matcher: https://docs.opencv.org/3.4/d3/da1/classcv_1_1BFMatcher.html

4) You will need to display the matches (for a single pair of images). Call the appropriate OpenCV drawMatch function.

4.3 Task 3: Construct a feature map

In this task, you will record a sequence of images that include a static calibration checkerboard in the lab. You will then work on developing a map of a set of tracked features, and identify the checkerboard pattern in your feature map. This task will most likely need to be completed outside of the lab time slot, and will make up a significant portion of your lab report.

1) While the robot is running start the ROS node **l3_data_saver.py**. It will save the images stream and robot pose matrices in a new folder, named l3_mapping data, in the current working directory. The pictures will be labeled 0-N and have corresponding poses in tf.txt. Open the pose.txt file to confirm that there are an equal number of poses and images.

2) Place the checkerboard in the environment near the Turtlebot, and record a sequence where the robot drives forward in a straight line approaching the checkerboard. Make sure the checkerboard is placed to the side of the path, but remains visible for a significant portion of the recording. Additionally, try to minimize the number of non-stationary objects in the camera's view.

3) Modify the feature detection, tracking and matching pipeline from part 2 to operate on stored images. You will need to extract all features that are consecutively tracked for more than 5 frames. Notes:

    a. `matches = self.bf.match(query_descriptors, train_descriptors)` returns a list of match objects that have the attributes trainIdx and queryIdx. These are ids are associated with the train descriptors and query descriptors respectively.
    b. `kp_new, des_new = self.orb.compute(gray, kp)` returns your keypoints and descriptors. Kp_new in this case will contain an attribute pt which is the tuple (x_pixel, y_pixel).

4) Using the recorded poses of the robot at each frame as the true state of the robot with no

covariance and the calibration information computed in the first part of the lab, estimate the position of each tracked feature in a common coordinate frame of your choosing. You will be using a gauss newton bundle adjustment method to estimate the location of each landmark.

a. You will start off by getting a rough estimate of where the landmarks are. Use the first and last measurement to triangulate a rough location of the landmark.
b. Next, you will derive your bundle adjustment algorithm. Write down the parameters that you wish to estimate and derive the measurement error function that you will be minimizing. Use the pixel coordinate model discussed in class. Consult gauss_newton.pdf on how to derive your non-linear gauss-newton optimization procedure.
c. Consider adding modifications such as minimum baselines between first and last measurements, outlier rejection such as RANSAC and other feature filtering measurements to improve the visibility of the target in the point cloud map. Store your point cloud using your preferred method.
d. The straight line motion should minimize pose errors due to wheel slip, but what other challenges do you observe when localizing feature landmarks in a visual odometry setting?

5) A ros node **l3_pt_cloud_node.py** has been provided to plot your pointcloud. You will need to change the pointcloud data loading part of the node to load your data. Additionally, your data must be in the shape Nx3. Start a roscore, run this node, and open rviz to view your data. Visualize the point cloud using rviz and look to see if you can identify the location of the checkerboard target.

5 Lab Report

This lab is a challenging exercise in using ROS, OpenCV and the Turtlebot, and may require significant independent investigation to create a working solution. The report for this lab is the only deliverable for the lab portion of the course, and needs only cover those aspects of your work that demonstrate understanding and the proper functioning of your code on real data. The report should be as concise as possible and include the following discussion and results:

1) Report your calibration parameters and images showing the rectified images of the calibration target.

2) Describe your feature detection, description and matching method, any parameters you selected or modifications you made to the provided instructions, and present 3 pairs of images and their matches. Discuss what you uncovered in terms of match quality for the 3 image pairs, and select the pairs in a way that demonstrates both strengths and weaknesses of feature-based front ends.

3) Present a derivation for your feature map construction that takes the robot pose as given and fused consecutive feature measurements into a feature location estimate. Present your resulting

pointcloud at 5 different timesteps along the trajectory of the robot, and identify the location of the checkerboard in each with some indicator.

4) Include your code for part 3 in an appendix, and submit a single PDF report.

6 Concluding Remarks

This lab exposes you to the implementation challenges of getting visual localization and mapping methods to work in the real world. In fact, the lab environment is still rather benign compared to the visual challenges that field robotics applications require. Mining, forestry, autonomous vehicles, oceanic vessels all require robust localization capabilities in far more severe conditions, and robotic vision remains a challenging open field to this day. We hope this lab gave you a little taste for how much fun these challenges can be.

7 Additional Resources

1. Introduction to ROS, ROB521 Handout, 2019.
2. Robotis e-Manual, http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/.
3. "SSH: Remote control your Raspberry Pi," The MagPi Magazine, https://www.raspberrypi.org/magpi/ssh-remote-control-raspberry-pi/.
4. Official ROS Website, https://www.ros.org/.
5. ROS Wiki, http://wiki.ros.org/ROS/Introduction.
6. Useful tutorials to run through from ROS Wiki, http://wiki.ros.org/ROS/Tutorials.
7. ROS Robot Programming Textbook, by the TurtleBot3 developers, http://www.pishrobot.com/wp-content/uploads/2018/02/ROS-robot-programming-book-by-turtlebo3-developers-EN.pdf.