Visual Navigation for Flying Robots
D. Cremers, X. Gao, V. Usenko
Summer Semester 2018

Computer Vision Group
Department of Informatics
Technical University of Munich

Exercise Sheet 1

Topic: Introduction to SLAM and Linux Programming

Submission deadline: Sunday, 22.04.2018, 23:59
Hand-in via email to visnav_ss2018@vision.in.tum.de

General Notice

The exercises should be done by yourself, but the final project should be done in teams of two to three students. We will use Ubuntu 16.04 in this lab course. It is already installed on the lab computers. If you want to use your own laptop, you will need to install Ubuntu by yourself.

Files related with the exercise will be placed in directories like 'paper/' or 'doc/'. Please read these materials before start answering the questions.

Exercise 1: What is SLAM

Survey papers are very useful when you want to quickly learn a new research area, especially for large topics like SLAM, which has almost 30 years history and a tremendous number of papers. Read the paper [1] (see papers/) and answer the following questions:

1. Why would a SLAM system need a map?
2. How can we apply SLAM technology into real-world applications?
3. Talk about the history of SLAM.

Exercise 2: Using cmake to manage SLAM projects written in C++

Cmake is a commonly used and convenient tool for building C++ programs under Linux. There are many libraries, such as OpenCV, g2o, Ceres, etc., that use cmake to build their projects. So, whether you are using someone else's library or writing your own one, you need to know the basics of cmake. Perhaps you have not heard of this tool before, but it does not matter, you can find many tutorials in website like `https://cmake.org/cmake-tutorial/` or `https://github.com/ttroy50/cmake-examples`. Please read them first and complete the following exercises.

1. Many SLAM projects are managed by cmake. Read the code of ORB-SLAM2 [1] or DSO[2], then show how these projects are organized. For example, for ORB-SLAM2, please answer how many libraries and executables are compiled during cmake process.

2. Write an executable program that uses ORB-SLAM2 or DSO to estimate the trajectory of your own camera. You can use your notebook camera or an USB camera which will make your hands more flexible. Or, if your computer happens to have no camera/you are running linux in virtual machine, please use the video in code/myvideo.mp4, which is captured in Theresienwiese by my cellphone.

   Since we haven't talked about any issues with OpenCV or images, I wrote a code/myslam.cpp file for you in this section (if you are using the recorded video, use code/myvideo.cpp instead). This code will open your own camera (or video), read the image, and hand it over to ORB-SLAM2. If you are using DSO, please write your own cpp file and it won't be very difficult. Since you have already understood the principles of cmake, please modify the cmake files in ORB/DSO to compile this cpp file.

Hints:

- You need to install the dependencies of the project. Please follow the instructions in readme or github.
- We haven't talked about camera calibration yet, so you currently have no way to calibrate your camera (unless you've learned by yourself). But it does not matter, we can also use a not so good calibration and just try it (fortunately ORB-SLAM2 is not sensitive to the calibration parameters). I have provided you with a myslam.yaml (myvideo.yaml), which is your hypothetical calibration parameter. Now, use this file to get ORB-SLAM2/DSO running.

Please show the running snapshots and discuss the effects of your program.

Exercise 4: Use Eigen library to handle geometry computation

Eigen `http://eigen.tuxfamily.org` is a highly efficient C++ library for linear algebra computation. In this section you need to write a program doing some geometry computations using Eigen.

1. Prove the Rodrigues' formula and verify it using Eigen. Suppose we have a rotation vector whose length is θ and direction is $\mathbf{n}$, then the corresponding rotation matrix $\mathbf{R}$ is:

$$\mathbf{R} = \cos\theta\mathbf{I} + (1 - \cos\theta)\mathbf{n}\mathbf{n}^T + \sin\theta\mathbf{n}^\wedge. \tag{1}$$

   Hints: use AngleAxis class in Eigen and convert it to rotation matrix.

[1] `https://github.com/raulmur/ORB_SLAM2`
[2] `https://github.com/JakobEngel/dso`

2. Suppose we have a quaternion $\mathbf{q}$. We denote its imaginary part as $\boldsymbol{\varepsilon}$ and the real part as η, then we can write any quaternion as: $\mathbf{q} = (\boldsymbol{\varepsilon}, \eta)$. Please show the dimension of $\boldsymbol{\varepsilon}$ and η.

   Furthermore, we define operator $^+$ and $^\oplus$:

$$\mathbf{q}^+ = \begin{bmatrix} \eta\mathbf{1} + \boldsymbol{\varepsilon}^\times & \boldsymbol{\varepsilon} \\ -\boldsymbol{\varepsilon}^\mathrm{T} & \eta \end{bmatrix}, \quad \mathbf{q}^\oplus = \begin{bmatrix} \eta\mathbf{1} - \boldsymbol{\varepsilon}^\times & \boldsymbol{\varepsilon} \\ -\boldsymbol{\varepsilon}^\mathrm{T} & \eta \end{bmatrix}, \tag{2}$$

   where $^\times$ means the same as $^\wedge$, i.e., the skew-symmetric matrix of $\boldsymbol{\varepsilon}$, and $\mathbf{1}$ is the identity matrix. Please show that for any unit quaternions $\mathbf{q}_1$, $\mathbf{q}_2$, we can denote the quaternion multiplication as matrix multiplication:

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{q}_1^+ \mathbf{q}_2 \tag{3}$$

   or

$$\mathbf{q}_1 \cdot \mathbf{q}_2 = \mathbf{q}_2^\oplus \mathbf{q}_1. \tag{4}$$

   By this way, please show how to convert an unit quaternion to rotation matrix, and verify it using Eigen.

3. Finally, lets do some numeric computations. Suppose we have robot 1 at $\mathbf{q}_1 = [0.55, 0.3, 0.2, 0.2], \mathbf{t}_1 = [0.7, 1.1, 0.2]^T$, robot 2 at $\mathbf{q}_2 = [-0.1, 0.3, -0.7, 0.2], \mathbf{t}_2 = [-0.1, 0.4, 0.8]^T$. Here, the first element of quaternion is the real part, and the $\mathbf{q}, \mathbf{t}$ is the rotation and translation from the world to the robot frame, which means when you multiply them with the any point's world coordinates, you will get its coordinates in the robot's frame. Now, the robot 1 observes a point whose coordinates are $\mathbf{p}_1 = [0.5, -0.1, 0.2]^T$ (in robot 1's frame), please compute its coordinates in robot 2's frame.

   Hints: (i) Please normalize the quaternion first. (ii) Please note the order of coefficients in quaternion. Some libraries put the real part at first and other put the real part last.

Submission instructions

A complete submission consists both of a PDF file with the solutions/answers to the questions on the exercise sheet and a ZIP file containing the source code that you used to solve the given problems. Note all names of your team members in the PDF file. Make sure that your ZIP file contains all files necessary to compile and run your code, but it should not contain any build files or binaries. Please submit your solution via email to `visnav_ss18@vision.in.tum.de`.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.