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 2

Topic: Lie group and Lie algebra

Submission deadline: Monday, 04.28.2015, 23:59 pm
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 14.04 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: Left Jacobian in $SE(3)$

We have shown the exponential map in $SO(3)$ in the lecture, now please derive the exponential map in $SE(3)$. Assume $\boldsymbol{\xi} = [\boldsymbol{\rho}, \boldsymbol{\phi}]^{\mathrm{T}} \in \mathfrak{se}(3)$, where $\boldsymbol{\rho}$ is the translation part and $\boldsymbol{\phi}$ is the rotation part. We know its exponential map is

$$\exp\left(\boldsymbol{\xi}^{\wedge}\right) = \begin{bmatrix} \sum_{n=0}^{\infty} \frac{1}{n!} (\boldsymbol{\phi}^{\wedge})^n & \sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\boldsymbol{\phi}^{\wedge})^n \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}. \tag{1}$$

Let $\boldsymbol{\phi} = \theta \boldsymbol{a}$, then we have

$$\sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\boldsymbol{\phi}^{\wedge})^n = \frac{\sin\theta}{\theta} I + \left(1 - \frac{\sin\theta}{\theta}\right) \boldsymbol{a}\boldsymbol{a}^T + \frac{1-\cos\theta}{\theta} \boldsymbol{a}^{\wedge} \triangleq \boldsymbol{J}. \tag{2}$$

Please prove this equation, and verify it using the Sophus library.

Hints: (i) use Taylor expansion and put the odd and even items together, just like what we do in $SO(3)$'s case. (ii) When verifying the result, you can choose an arbitrary vector as a $\mathfrak{se}(3)$ element, use the exp() function in Sophus/se3.hpp, and then show the exponential map result is same as the formula we've shown.

Exercise 2: Compare trajectories

When we want to evaluate SLAM systems, it is always useful to compare the estimated trajectory to the ground-truth. In this section, I will give to two files which contain an estimated and a ground-truth trajectory. Please complete the following tasks:

1. Read and plot the trajectories. The trajectory files are stored in code/ground-truth.txt and code/estimate.txt. Each line in the file is a pose with the format

$$[t, t_x, t_y, t_z, q_x, q_y, q_z, q_w],$$

where t is the time stamp, t_x, t_y, t_z is the translation part, and q_x, q_y, q_z, q_w is the rotation part (q_w is the real part of the quaternion). Here the $\boldsymbol{q}, \boldsymbol{t}$ represent the transform from the camera to the world, i.e. the $\boldsymbol{T}_{wc}$. Please write a program (see code/draw_trajectory.cpp) that reads the trajectory files and plot them in a Pangolin window.

2. We can further compute the estimated error of the trajectory. Indicators that are commonly used here are ATE (Average Translational Error), RPE (Relative Pose error) or RMSE (Root mean squared error). Let $\boldsymbol{T}_{g,k}$ be the ground-truth trajectory and $\boldsymbol{T}_{e,k}$ be the estimated one, where $k = 1, \ldots, n$, then the ATE can be written as:

$$\text{ATE} = \frac{1}{n} \sum_{k=1}^{n} \| \text{trans}(\boldsymbol{T}_{g,k}) - \text{trans}\left(\boldsymbol{T}_{e,k}\right) \|_2^2. \tag{3}$$

where the trans() function takes the translation part of the transform matrix. Of course, this error indicates only the translational part. We can also define the RMSE of alignment error as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{k=1}^{n} \| \log(\boldsymbol{T}_{g,k}^{-1} \boldsymbol{T}_{e,k})^{\vee} \|_2^2}. \tag{4}$$

Using these definitions, please compute the ATE and RMSE of alignment error of the given two trajectories.

Exercise 3: Images, camera intrinsic and extrinsic

We have talked about the intrinsic, extrinsic and distortion model in camera calibration. Please complete the following tasks using the knowledge you've learned in the course.

1. Image undistortion. The code/test.png is a distorted image from EuRoC dataset. The radial and tangential distortion model is:

$$\begin{cases} x_{\text{distorted}} = x\left(1 + k_1 r^2 + k_2 r^4\right) + 2p_1 xy + p_2\left(r^2 + 2x^2\right) \\ y_{\text{distorted}} = y\left(1 + k_1 r^2 + k_2 r^4\right) + p_1\left(r^2 + 2y^2\right) + 2p_2 xy \end{cases} \tag{5}$$

Now given the distortion parameters:

$k_1 = -0.28340811$, $k_2 = 0.07395907$, $p_1 = 0.00019359$, $p_2 = 1.76187114e-05$

and camera intrinsics:

$$f_x = 458.654,\ f_y = 457.296,\ c_x = 367.215,\ c_y = 248.375.$$

Please write a program (using code/undistort_image.cpp) to recover the undistorted image.

Note: Please don't use OpenCV's undistortion functions, you need to do this computation by yourself. However, you can use OpenCV function to check if your code is right or not. And after undistortion, the straight lines in 3D should look straight in the image.

2. Stereo vision. Stereo cameras can use the disparity to compute the depth of pixels. Here we give you two images: code/left.png and code/right.png, and also the disparity image: code/disparity.png. The intrinsics of stereo camera are:

$$f_x = 718.856,\ f_y = 718.856,\ c_x = 607.1928,\ c_y = 185.2157$$

and the baseline is $b = 0.573$ m. The pixel values in the disparity image is $d = u_L - u_R$, so for any pixel in the left image (u_L, v_L), $(u_L - d, v_L)$ should be its corresponding point in the right image.

Please recover the point cloud in the left eye using the disparity image, and then show it in a Pangolin window. You can use the code in code/disparity.cpp.

3. RGB-D vision. The code/RGBD1.png - code/RGBD5.png are five images captured by an RGB-D camera, and code/depth1.pgm - code/depth5.pgm are the corresponding depth images. The intrinsics of the RGB-D camera are given in the code, and the extrinsics are stored in code/RGBD-extrinsic.txt in the same format as exercise 2. Now you can build a point cloud map of the whole environment by computing the position of each pixel in the world frame. Please complete the code/build-map.cpp to build a point cloud map and display it in the Pangolin.

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_ss2018@vision.in.tum.de`.