Exercise Sheet 1

Topic: Introduction to SLAM and Lie Groups

Yun-Jin Li

November 4, 2022

Par1: Setup, Test Submission, and CMake

The test submission is already uploaded onto the server, and it can run successfully.

1. We append the variable CMAKE_MODULE_PATH with the directory /cmake_modules which is in the source directory.

2. We specify C++14 version, and prevent the compiler from falling back to previous standard if it doesn't support C++14. Furthermore, we disable the usage of compiler-specific extensions. By default, on a Linux system, CMake passes -std=gnu++14 to GCC, but we would like to build with -std=c++14 to prevent non-portable compiler extensions.

3. CMake has 4 build types: debug (Debug), release (Release), release with debug information (RelWithDebInfo), and minimum size release (MinSizeRel) [1]. Therefore, this section basically specifies the properties for each build type.

   - Debug: Optimization level = 0 (O0) [2], Default compile options for DEBUG build type (-g), set the option EIGEN_INITIALIZE_MATRICES_BY_NAN as True to make all entries of newly constructed matrices and arrays from Eigen initialized to NaN [3].
   - RelWithDebInfo: Optimization level = 3 (O3) [2], Default compile options for RelWithDebInfo build type (-g), Choose option with not using debugging information (-DNDEBUG) [4], set the option EIGEN_INITIALIZE_MATRICES_BY_NAN as True to make all entries of newly constructed matrices and arrays from Eigen initialized to NaN [3].
   - Release: Optimization level = 3 (O3) [2], Choose option with not using debugging information (-DNDEBUG).

   CMAKE_CXX_FLAGS is used to add flags for all C++ targets. We set the following flags in our CMakeLists.txt.

   - Set the maximum number of template instantiation notes for a single warning or error to 0. (-ftemplate-backtrace-limit =0)
   - Set warning level -Wall, -Wextra, and our self-defined variable EXTRA_WARNING_FLAGS. Details could be found here [5].
   - Specify the CPU type of our machine by -march.

4. Add our executable target called ”calibration” to be built from the source file calibration.cpp and specify libraries Ceres::ceres, pangolin, TBB that would be used by the target ”calibration”.

Part 2: SO(3) and SE(3) Lie groups

$$SE(3) \equiv \left\{ g = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \,\middle|\, R \in SO(3),\ T \in \mathbb{R}^3 \right\}$$

$\hat{\xi} \in se(3)$ and $\xi = \begin{bmatrix} v \\ w \end{bmatrix}$ — v: translation, w: rotation

$$g: \mathbb{R} \to SE(3) \quad g(t) = \begin{bmatrix} R(t) & T(t) \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4}$$

$$\dot{g}(t) = \begin{bmatrix} \dot{R}(t) & \dot{T}(t) \\ 0 & 0 \end{bmatrix}$$

$$g(t)^{-1} = \begin{bmatrix} R^T(t) & -R^T(t)T(t) \\ 0 & 1 \end{bmatrix}$$

$$\dot{g}(t)g(t)^{-1} = \begin{bmatrix} \dot{R}(t)R^T(t) & -\dot{R}(t)R^T(t)T(t) + \dot{T}(t) \\ 0 & 0 \end{bmatrix} \quad \text{— (1)}$$

We know that from $RR^T = I$

$$\Rightarrow \frac{d}{dt}(RR^T) = 0$$
$$\Rightarrow \dot{R}R^T = -R\dot{R}^T$$
$$\Rightarrow \dot{R}R^T = -(\dot{R}R^T)^T$$

so $\dot{R}R^T$ is a skew symmetric matrix

we define it as $\hat{w} = \dot{R}R^T \in so(3)$, $w \in \mathbb{R}^3$

And we define a vector $v(t) = -\hat{w}(t)T(t) + \dot{T}(t)$

We could eventually rewrite (1) as

$$\dot{g}(t)g(t)^{-1} = \begin{bmatrix} \hat{w}(t) & v(t) \\ 0 & 0 \end{bmatrix} \equiv \hat{\xi}(t) \quad \text{— (2)}$$

If we multiply (2) with $g(t)$ on the right side we have:

$$\dot{g}(t)g(t)^{-1}g(t) = \hat{\xi}(t)g(t)$$

$\Rightarrow \dot{g}(t) = \hat{\xi}(t)g(t)$, which basically means that $\hat{\xi}$ is the tangent vector along curve $g(t)$

we define lie algebra $se(3) \equiv \left\{ \hat{\xi} = \begin{bmatrix} \hat{w} & v \\ 0 & 0 \end{bmatrix} \,\middle|\, \hat{w} \in so(3), v \in \mathbb{R}^3 \right\}$

Write our differential equation with constant twist coordinate $\xi = \begin{bmatrix} v \\ w \end{bmatrix}$

$$\begin{cases} \dot{g}(t) = \hat{\xi}\, g(t), & \hat{\xi} = \text{constant} \\ g(0) = I \end{cases}$$

Using Taylor expansion, we have:

with $\Delta t = t$
$$g(t) = g(0) + \hat{\xi} g(0) t + \frac{1}{2!} \xi^2 g(0) t^2 + \cdots$$
$$= I + \hat{\xi} t + \frac{1}{2!} \xi^2 t^2 + \cdots$$
$$= \sum_{n=0}^{\infty} \frac{(\hat{\xi} t)^n}{n!}$$

And this is exactly equel to the power serie expansion of an exponential function

$$\Rightarrow g(t) = e^{\hat{\xi} t}$$
$$\Rightarrow \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} = e^{\hat{\xi}} = \begin{bmatrix} \sum_{n=0}^{\infty} \frac{\hat{\omega}^n}{n!} & \sum_{n=0}^{\infty} \frac{\hat{\omega}^n}{(n+1)!} v \\ 0 & 1 \end{bmatrix}$$

$$\frac{1}{1!} \xi^1 = \frac{1}{1!} \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}$$
$$\frac{1}{2!} \xi^2 = \frac{1}{2!} \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \hat{\omega}^2 & \hat{\omega} v \\ 0 & 0 \end{bmatrix}$$
$$\frac{1}{3!} \xi^3 = \frac{1}{3!} \begin{bmatrix} \hat{\omega}^3 & \hat{\omega}^2 v \\ 0 & 0 \end{bmatrix}$$
$$\vdots$$

based on the order we could conclude that

$$\sum_{n=0}^{\infty} \frac{\hat{\omega}^n}{(n+1)!} = I + \frac{\hat{\omega}}{2!} + \frac{\hat{\omega}^2}{3!} + \cdots \quad - (3)$$

Let $\theta = \|\omega\|$

and $\bar{\omega} = \frac{\omega}{\|\omega\|} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix}$ when $\omega_1^2 + \omega_2^2 + \omega_3^2 = 1$

$$\hat{\bar{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$
$$\hat{\bar{\omega}}^2 = \begin{bmatrix} -\omega_3^2 - \omega_2^2 & \omega_1 \omega_2 & \omega_1 \omega_3 \\ \omega_1 \omega_2 & -\omega_3^2 - \omega_1 & \omega_2 \omega_3 \\ \omega_1 \omega_3 & \omega_2 \omega_3 & -\omega_2^2 - \omega_1^2 \end{bmatrix} = \bar{\omega} \bar{\omega}^T - I$$
$$\hat{\bar{\omega}}^3 = -\hat{\bar{\omega}}$$
$$\hat{\bar{\omega}}^4 = -\hat{\bar{\omega}}^2$$

The implementation has also been pushed to the server, and the build test has passed as well.

So, ③ could be rearranged as follow:

$$I + \left(\frac{\theta}{2!} - \frac{\theta^3}{4!} + \frac{\theta^5}{6!}\right)\hat{\omega} + \left(\frac{\theta^2}{3!} - \frac{\theta^4}{5!} \cdots \right)\hat{\omega}^2$$

We know that $\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots$

$$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots$$

$$= I + \underbrace{\left(\frac{-\theta^2}{2!} + \frac{\theta^4}{4!} - \cdots\right)}_{\cos\theta - 1}\frac{\hat{\omega}}{(-\theta)} + \underbrace{\left(\frac{-\theta^3}{3!} + \frac{\theta^5}{5!} - \cdots\right)}_{\sin\theta - \theta}\frac{\hat{\omega}^2}{(-\theta)}$$

$$= I + (\cos\theta - 1)\frac{-\hat{\omega}}{\theta^2} + (\sin\theta - \theta)\frac{-\hat{\omega}^2}{\theta^3}$$

$$= I + \frac{(1 - \cos\theta)}{\theta^2}\hat{\omega} + \frac{\theta - \sin\theta}{\theta^3}\hat{\omega}^2$$

#

Part 3: What is SLAM?

1. In order to build a globally consistent representation of the environment, we not only utilize self-motion measurements but also loop closures. Without loop closures, SLAM reduces to only odometry which is easy to drift. Furthermore, the representation we build would simply be an infinite corridor. However, with the presence of a map, we can model the true topological structure of the environment which means we can know what is the shortest path between the start and the goal position when we're for instance trying to solve some path planning tasks.

2. Typically, SLAM is separated into two parts, front-end and back-end. Front-end is basically in charge of extracting meaningful data from sensors such as a lidar or a camera. This would include something like feature extraction and data association. Once we complete this step, we can use some probabilistic approach such as MAP estimation to do either loop closures or verification and that is what we called the back-end. By following this pipeline, we can accomplish so many real-world applications such as indoor mobile robot navigation, etc.

3. In the classical age (1986 - 2004), they introduced the main probabilistic formulations for SLAM, such as Extended Kalman Filters, Particle Filters, etc. Later, the so-called algorithmic-analysis age (2004 - 2015) came and the fundamental properties of SLAM were investigated, such as observability, convergence, and consistency. Moreover, the main open-source SLAM libraries were also developed in this period. The paper particularly pointed out that we're currently in the third era of SLAM where robust performance, high-level understanding, resource awareness, and task-driven perception are the key requirements.

References

[1] The build types of cmake. [Online]. Available: https://cmake.org/cmake/help/latest/variable/CMAKE_BUILD_TYPE.html

[2] Options that control optimization. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

[3] Eigen - preprocessor directives. [Online]. Available: https://eigen.tuxfamily.org/dox/TopicPreprocessorDirectives.html

[4] C programming/assert.h. [Online]. Available: https://en.wikibooks.org/wiki/C_Programming/assert.h#:~:text=The%20macro%20NDEBUG%20denotes%20not,expression%20it%20tests%20is%20false.

[5] Warning options. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html