Vision Based Navigation Assignment 1

Erkam Uyanik (03709181)

09-11-2019

Exercise 1

1. Why would a SLAM system need a map?

   First, a map may be needed for other tasks, e.g. path planning.

   Second, having a map allows limiting accumulated error in state estimation by loop closing.

2. How can we apply SLAM technology into real-world applications?

   SLAM systems typically have two parts; frontend where feature extraction and data assocation is done, and backend where MAP estimation is done. Frontend component abstracts sensor data into a form usable by MAP estimation. Backend component estimates state variables usually using MAP and factor graphs.

   How to apply SLAM depends on the task we want to solve, available sensors and computational resources, and expected criteria such as accuracy and latency.

   There are open source SLAM frameworks such as ORB-SLAM which makes it possible to easily integrate it to a real world problem. Other than many applications through years, autonomous vehicles are the main research area today for the application of SLAM with the help of 3D LIDAR sensor technology.

3. Describe the history of SLAM.

   SLAM has a history of roughly 35 years as of 2019. Period of 1986-2004 is called as *classical age*. During the classical age, main probabilistic formulations are introduced such as Extended Kalman Filters. Period of 2004-2015 is called *algorithmic analysis age* where fundamental SLAM properties are studied. The paper argues that we are entering a new era, namely *robust perception age* where robust performance, high level understanding of the environment, resource awareness, and task driven perception are key requirements. New sensor technologies and advancements in areas such as deep learning are also important factors in advancements in SLAM for this era.

Exercise 2

```
set(CMAKE_MODULE_PATH "${
    ↪ CMAKE_CURRENT_SOURCE_DIR}/cmake_modules/
    ↪ " ${CMAKE_MODULE_PATH})
```

This command sets multiple values to *CMAKE_MODULE_PATH*. By reassigning *$CMAKE_MODULE_PATH* it appends existing value of this variable with the other value.

```
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_EXTENSIONS OFF)
```

It sets C++ standard to version 11, and makes it required to prevent a fall back to an older version. Last line prevents use of *-std=gnu++11* flag instead of *-std=c++11* for version 11.

```
set(CMAKE_CXX_FLAGS_DEBUG  "-O0 -g -
    ↪ DEIGEN_INITIALIZE_MATRICES_BY_NAN")
```

This command updates flags for debug mode.

- -O0: turn off optimization
- -g: build with debugging symbols
- -DEIGEN_INITIALIZE_MATRICES_BY_NAN: initialize Eigen library matrices with *nan* values

```
set(CMAKE_CXX_FLAGS_RELWITHDEBINFO "-O3 -g -
    ↪ DEIGEN_INITIALIZE_MATRICES_BY_NAN")
```

This command updates flags for *RelWithDebInfo* mode (release with debug info).

- -O3: high level optimization
- -g: build with debugging symbols
- -DEIGEN_INITIALIZE_MATRICES_BY_NAN: initialize Eigen library matrices with *nan* values

```
set(CMAKE_CXX_FLAGS_RELEASE "-O3 -DNDEBUG")
```

This command updates flags for *release* mode.

- -O3: high level optimization
- -DNDEBUG: disable assertions

```
SET(CMAKE_CXX_FLAGS " -ftemplate-backtrace-
    ↪ limit=0 -Wall -Wextra ${
    ↪ EXTRA_WARNING_FLAGS} -march=${CXX_MARCH}
    ↪ ${CMAKE_CXX_FLAGS}")
```

- -ftemplate-backtrace-limit=0: do not instantiate any template for a warning/error
- -Wall: turns on (almost) all warnings
- -Wextra: turns on extra warnings
- *$EXTRA_WARNING_FLAGS*: enable flags set to this variable
- *$CMAKE_CXX_FLAGS*: append all of these to existing flags
- -march=$CXX_MARCH: sets target CPU for the compiler

```
add_executable(calibration src/calibration.cpp
    ↪ )
target_link_libraries(calibration ceres ${
    ↪ Pangolin_LIBRARIES} ${TBB_LIBRARIES
})
```

This creates an executable for *calibration* starting from *src calibration.cpp*. It also specifies libraries this executable depends on, i.e. ceres, pangolin, and threading building blocks.

Exercise 3

$$
\begin{aligned}
\sum_{n=0}\infty \frac{1}{(n+1)!}(\theta a^{\wedge})^n &= I + \frac{1}{2!}\theta a^{\wedge} + \frac{1}{3!}\theta^2(a^{\wedge})^2 + \frac{1}{4!}\theta^3(a^{\wedge})^3 + \frac{1}{5!}\theta^4(a^{\wedge})^4 + \dots \\
&= (aa^T - (a^{\wedge})^2) + \frac{1}{2!}\theta a^{\wedge} + \frac{1}{3!}\theta^2(a^{\wedge})^2 - \frac{1}{4!}\theta^3 a^{\wedge} - \frac{1}{5!}\theta^4(a^{\wedge})^2 + \frac{1}{6!}\theta^5 a^{\wedge} + \dots \\
&= aa^T + a^{\wedge}\left(\frac{1}{2!}\theta - \frac{1}{4!}\theta^3 + \frac{1}{6!}\theta^5 + \dots\right) - (a^{\wedge})^2\left(1 - \frac{1}{3!}\theta^2 + \frac{1}{5!}\theta^4 - \cdots\right) \\
&= aa^T + a^{\wedge}\left(\frac{1-\cos\theta}{\theta}\right) - (a^{\wedge})^2\left(\frac{\sin\theta}{\theta}\right) \\
&= (a^{\wedge})^2 + I + a^{\wedge}\left(\frac{1-\cos\theta}{\theta}\right) - (a^{\wedge})^2\left(\frac{\sin\theta}{\theta}\right) \\
&= (a^{\wedge})^2\left(1 - \frac{\sin\theta}{\theta}\right) + I + a^{\wedge}\left(\frac{1-\cos\theta}{\theta}\right) \\
&= \left(1 - \frac{\sin\theta}{\theta}\right)\left(aa^T - I\right) + I + a^{\wedge}\left(\frac{1-\cos\theta}{\theta}\right) \\
&= \left(1 - \frac{\sin\theta}{\theta}\right)aa^T + I\frac{\sin\theta}{\theta} + a^{\wedge}\left(\frac{1-\cos\theta}{\theta}\right)
\end{aligned}
$$