**Exercise 1**

Part 1 :

1. This line of code ensures that CMake can find any additional CMake modules needed by the project at the path ${CMAKE_CURRENT_SOURCE_DIR}/cmake_modules/ and adds the new directory to the search path while keeping any previously set directories.

2. First line of code sets the language standard to C++14. Second line indicates to that the selected C++ standard i.e 14, is required for the project to compile successfully. Third lie tells the C++ compiler to not enable any non-standard language extensions.

3. First line specifies a set of flags to be used by the C++ compiler when building in debug mode. Second line specifies a set of flags to be used by the C++ compiler when building in Release with Debug Information mode. Third line specifies a set of flags to be used by the C++ compiler when building in Release mode. Fourth line specifies a set of flags to be used by the C++ compiler when building the code.

4. First line creates a new target in the build system and specifies the source files needed to build it. In creates an executable ‘calibration’ from the file ‘src / calibration . cpp ’. Second line specifies the libraries that need to be linked with the target executable. So the libraries ceres, pangolin, and TBB will be linked to the executable ‘calibration’.

Page No.
Date

Part 2:

To Prove:

$$\sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\hat{w})^n = I + \frac{1-\cos(\theta)}{\theta^2}\hat{w} + \frac{\theta - \sin(\theta)}{\theta^3}(\hat{w})^2 = J$$

It is given that we can take $\theta = \|w\|$

So, let $v = \frac{w}{\|w\|} = \frac{w}{\theta}$

So that $w = v\theta$

Therefore,

$$\sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\hat{w})^n = \sum_{n=0}^{\infty} \frac{1}{(n+1)!} (\hat{v}\theta)^n$$

Expanding this equation,

$$\Rightarrow I + \frac{\hat{v}\theta}{2!} + \frac{(\hat{v}\theta)^2}{3!} + \frac{(\hat{v}\theta)^3}{4!} + \cdots\cdots$$

grouping even & odd terms,

$$\Rightarrow I + \left(\frac{\hat{v}\theta}{2!} + \frac{(\hat{v}\theta)^3}{4!} + \frac{(\hat{v}\theta)^5}{6!} + \cdots\right) + \left(\frac{(\hat{v}\theta)^2}{3!} + \frac{(\hat{v}\theta)^4}{5!} + \frac{(\hat{v}\theta)^6}{7!} + \cdots\right)$$

It can be inferenced that $\hat{v}^2 = vv^T - I$, $\hat{v}^3 = -\hat{v}$, $\hat{v}^4 = -\hat{v}^2$, $\hat{v}^5 = \hat{v}$ etc Since it is a skew symmetric matrix.

$$\Rightarrow I + \left(\frac{\hat{v}\theta}{2!} + \frac{\hat{v}^3\theta^3}{4!} + \frac{\hat{v}^5\theta^5}{6!} + \cdots\right) + \left(\frac{\hat{v}^2\theta^2}{3!} + \frac{\hat{v}^4\theta^4}{5!} + \frac{\hat{v}^6\theta^6}{7!} + \cdots\right)$$

$$\Rightarrow I + \left(\frac{\hat{v}\theta}{2!} - \frac{\hat{v}\theta^3}{4!} + \frac{\hat{v}\theta^5}{6!} - \cdots\right) + \left(\frac{\hat{v}^2\theta^2}{3!} - \frac{\hat{v}^2\theta^4}{5!} + \frac{\hat{v}^2\theta^6}{7!} - \cdots\right)$$

Page No.
Date

$$\Rightarrow I + \left(\frac{\theta}{2!} - \frac{\theta^3}{4!} + \frac{\theta^5}{6!} + \dots\right)\hat{v} + \left(\frac{\theta^2}{3!} - \frac{\theta^4}{5!} + \frac{\theta^6}{7!} + \dots\right)\hat{v}^2$$

$$\Rightarrow I + \left(\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \frac{\theta^6}{6!} + \dots\right)\frac{\hat{v}^2}{\theta} + \left(\frac{\theta^3}{3!} - \frac{\theta^5}{5!} + \frac{\theta^7}{7!} + \dots\right)\frac{\hat{v}^2}{\theta}$$

Now we know that,

$\cos\theta = 1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} + \frac{\theta^6}{6!} + \dots$ and $\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} + \frac{\theta^7}{7!}$

So, $\frac{\theta^2}{2!} - \frac{\theta^4}{4!} + \dots = 1 - \cos\theta$ and $\frac{\theta^3}{3!} - \frac{\theta^5}{5!} + \dots = \theta - \sin\theta$

Applying these,

$$\Rightarrow I + (1 - \cos\theta)\frac{\hat{v}}{\theta} + (\theta - \sin\theta)\frac{\hat{v}^2}{\theta}$$

Substituting value of $v$,

$$\Rightarrow I + (1 - \cos\theta)\frac{\vec{w}}{\theta^2} + (\theta - \sin\theta)\frac{\vec{w}^2}{\theta^3} = \underline{\underline{T}}$$

Part 3:

1. The need to use a map of the environment is twofold. First, the map is often required to support other tasks like a map can inform path planning or provide an intuitive visualization for a human operator. Second, the map allows limiting the error committed in estimating the state of the robot. In the absence of a map, the system could drift from its path over time. The map serves as a reference frame that the robot can use to match its sensor measurements to features in the environment. It helps a SLAM system in areas like localization, planning, loop closure etc. Basically, it enables the system to estimate its position and navigate through the environment.

2. In order for SLAM algorithms to be applied to real world applications, it should satisfy some factors like 1) Robustness : SLAM systems need to be able to handle noisy sensor data, dynamic environments, and other sources of uncertainty. 2) Efficiency : Real-world applications require SLAM systems that can operate in real-time and in resource-constrained environments. 3) Scalability : Real-world applications often involve large-scale environments that require SLAM systems that can scale to handle large amounts of data. 4) SLAM systems need to be able to integrate with other systems and technologies, such as sensors, controllers, and software frameworks.
   So given these factors are fulfilled, SLAM algorithms can be applied to real world applications such as robotic navigation, autonomous driving, augmented reality etc.

3. The early algorithms, i.e. in the early 2000s, were often based on Kalman filters or particle filters and relied on range sensors such as sonar or laser range finders.
   After that starting from 2011, SLAM research began to focus more on visual sensors, such as cameras, which allowed for more detailed and informative data to be captured. This led to the development of feature-based SLAM algorithms, which used visual features such as corners and edges to estimate the robot's position and orientation.Graph-based SLAM algorithms were also developed, which represented the environment as a graph and used optimization techniques to estimate the robot's pose and the map of the environment. These algorithms were able to handle loop closures.
   In recent years, SLAM research has continued to evolve by advances in machine learning and computer vision. There has been an emergence of deep learning-based SLAM algorithms, which use neural networks to learn mappings between sensor data and maps.