Автономная некоммерческая организация
высшего образования
«Университет Иннополис»

Autonomous noncommercial organization
of higher education
“Innopolis University”

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ПО НАПРАВЛЕНИЮ ПОДГОТОВКИ
09.03.01 – «ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»

GRADUATE THESIS FIELD OF STUDY
09.03.01 – «COMPUTER SCIENCE»

НАПРАВЛЕННОСТЬ (ПРОФИЛЬ) ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ
«ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА»
AREA OF SPECIALIZATION / ACADEMIC PROGRAM TITLE:
«COMPUTER SCIENCE»

Тема	**Сравнение нелинейных оптимизационных методов для одновременной локализации и картографии**
Topic	**Compare nonlinear optimization techniques for visual simultaneous localization and mapping system**

Работу выполнил / Thesis is executed by	**Макаев Борис Олегович** **Makaev Boris Olegovich**	подпись / signature
Научный руководитель / Thesis supervisor	**Афанасьев Илья Михайлович** **Afanasyev Ilya Mikhailovich**	подпись / signature

Иннополис, Innopolis, 2019

Contents

Abstract

Simultaneous Localization and Mapping (SLAM) problem currently is massively popular; its applications range from Augmented Reality to Autonomous Car. The improvement of computational power is responsible for the rapid development of SLAM systems, but one problem still remains important: how to improve the computational efficiency of SLAM? Years of research has shown that part of the SLAM could be formulated through several nonlinear optimization problems and the choice of the method for solving these problems can greatly impact on the performance of the SLAM. This study aims to analyze the role of nonlinear optimization methods in SLAM. More specifically, the question arises: How the choice of specific methods for solving optimization problems in SLAM impacts on the performance of the SLAM system? Simultaneous Localization and Mapping consists of two main tasks: (1) construct and update the map of the environment (e.g. point cloud), (2) simultaneously track the location of the agent relative to the built map.

ORB-SLAM library was chosen as the specific system of interest; it uses g2o optimization framework for solving five optimization problems with Levenberg-Marquardt method. To assess the performance of different configurations of nonlinear optimization problems for ORB-SLAM the application with Graphical User Interface (GUI) was implemented; the application allows a user to specify the configuration of optimization problems in ORB-SLAM and launch the experiment that will produce results that can evaluate the performance of current configuration. Also, the functionality of ORB-SLAM was extended with the same optimization problems formulated using Ceres solver optimization framework.

Results present the comparison of different experiments using Comparison application; they show that different nonlinear optimization, methods, in fact, impact on the performance of the overall SLAM system. An interesting observation was made that among all experiments in the context of experiments that were executed the fastest in terms of mean/median tracking time is the most inaccurate in terms of absolute pose error (error between generated and reference trajectory).

Chapter 1

Introduction

Nowadays, computer vision (CV) algorithms are used in a great number of areas of robotics, such as autonomous cars, umanned aerial vehicles (UAV). The number of such applications that uses computer vision continue to rapidly grow under the conditions of enormous work that has been done by a great number of researches. A lot of algorithms like Simultaneous Localization and Mapping (SLAM) and Visual Odometry algorithms were implemented in order to allow robotic systems to perform visual tracking, estimate their position and trajectory relative to the world, build 3D map of the location they are in; data for these algorithms come from sensors like camera, inertial measurement units (accelerometers, gyroscopes, etc.). But the big problem of how to increase the performance of such algorithms arise; they carry out heavy mathematical computation and it is really important to choose the right methods to solve particular tasks. Particularly part of the problems of SLAM can be formulated using nonlinear optimization; such problems include minimization of reprojection error, bundle adjustment, etc. Different methods for solving nonlinear optimization problems can perform differently depending on the type of the task it tries to solve; thus, it is important to compare different optimization methods in order to pick the most efficient.

This thesis tries to establish the connection between the choice of nonlinear optimization methods in SLAM and overall performance of the SLAM algorithms. Particularly ORB-SLAM[1, 2] was chosen to be the base SLAM system which implements its nonlinear optimization problems using graph optimization framework (g2o). g2o optimization framework implements three nonlinear optimization methods: Levenberg-Marquardt, Gauss-Newton, and Dogleg; this thesis tries to extend the choice of optimization methods for ORB-SLAM via implementing optimization problems using Ceres solver optimization framework. The specific question could be then answered: does the choice of the optimization framework that implements solving of optimization problems impacts on the performance of ORB-SLAM? Also this thesis presents the implementation of Comparison application which provides GUI interface to interact with configuration of optimization problems for ORB-SLAM; it will also save the results

of the experiments (metrics that will evaluate the overall performance of the SLAM system, such as mean tracking time, median tracking time, and absolute pose error between trajectory generated by SLAM algorithm and reference trajectory (ground truth that was captured by the creators of datasets)); later when the results of the experiments are solved one could visualize and compare different results. Such Comparison application could be used then to try to answer on questions defined above.

Chapter 2 will establish the ground to work on in later chapters by defining important concepts and reviewing related literature. Chapter 3 will describe how we will be comparing ORB-SLAM configuration's with different nonlinear optimization methods; this chapter will introduce the ORB-SLAM extensions - implementation of nonlinear optimization problems using Ceres solver optimization framework; also this chapter will introduce Comparison application written in Qt that allows the user to set the configuration for ORB-SLAM, save the performance results and visualize these results. Chapter 4 will describe the practicalities and implementations of the concepts that were described in 3. Chapter 5 will evaluate the work of Comparison application and then will compare results of different experiments which will show how the choice of nonlinear optimization methods impacts the performance of ORB-SLAM. Chapter 6 will finish this thesis, summarize results and discuss future works that could be done in order to improve this thesis late.

Chapter 2

Literature Review

This chapter describes the results of important research works done in the field of Nonlinear optimization and SLAM methods. Especially, it is needed to establish a full understanding of what the full system that uses SLAM algorithm consists of. As it was stated the central topic of this thesis is comparing nonlinear optimization techniques in visual SLAM. We will find what nonlinear optimization techniques are used in existing SLAM systems and then prepare the ground for comparative analysis of different nonlinear optimization methods.

2.1 Optimization Problems

The task of optimization is present in many fields of human science and business activities. The goal of solving the optimization problem is to improve certain objective given parameters of the system. The system itself could be related to a business model where one wants to maximize profit; data analysis is also the kind of optimization problem that builds a model from data while minimizing specific error; a good example of data analysis optimization problem is regression, i.e. fitting function to data.

2.1.1 Problem Definition

The general optimization problem could be described with

$$\min_{x \in \mathbb{R}^n} f_0(x), \tag{2.1}$$

where $\mathbf{x}$ is vector design parameters (also called optimization variables) and $f_0(x)$ is the objective function that we want to minimize. We can convert minimization problem to maximization by setting $f_0(x) = -g(x)$, where $g(x)$ is some function that we want to maximize.

Usually, a minimization problem (2.1) is supplied with constraints that define the allowable values for optimization variables. There are two types of constraints: inequality constraints of form $f_i(x) \leq 0$ for $i = 1, 2, ..., m$ and

equality constraints of form $h(x) = 0$. Unconstrained optimization methods: like nonlinear equations, nonlinear least squares, global optimization. Constrained methods include Nonlinear Programming, Linearly Constrained Linear Programming, and Quadratic programming. [3]

It can be further stated that optimization problems can be divided into two big classes: linear programming problems and nonlinear programming problems. When all the constraints and the objective function are linear functions, the problem is *linear programming* problem. On the other hand, when at least one objective function or constraint function is nonlinear, the problem is *nonlinear programming* problem. Both classes of problems try to solve their corresponding modeled tasks, e.g. nonlinear problems arise from natural sciences like physics, chemistry, etc. and linear problems arise in financial areas like economics, management, etc.

Another fundamental optimization property that should be discussed is convexity. Function f is convex if for any two points in function domain there is the straight line that connects these points without intersecting the function. Convexity property can be found in many practical problems and simplifies the solution of the optimization task. One of the greatest implications of this property is that any local solution is a global solution.

Implementation of optimization problem solvers usually has iterative nature; given an initial guess, the algorithm then improves estimation until termination. Stephen Wright and Jorge Nocedal [3] propose three main properties that good optimization algorithm should have. The algorithm should be *robust* to different problems of the same nature; algorithm should be *efficient* in terms of computational consumption; algorithm should be able to identify some *accurate* solution.

The figure 2.1 shows one of the possible taxonomies of optimization methods. We are interested in particularly nonlinear unconstrained optimization methods.

2.1.2 Linear Optimization

Linear optimization or linear programming have a linear objective function and linear constraints. Usually, the problem is formulated like this:

$$\min c^T x, \text{ subject to } Ax = b, x \geq 0, \tag{2.2}$$

where $c \in \mathbb{R}^n$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $A \in \mathbb{R}^m \times \mathbb{R}^n$

The modern era in optimization started in the late 1940s when Dantzig developed the first method for solving a linear optimization problem. This method is called *Simplex method* and since then it has been refined and improved for the needs of the applications. With the rise of the computers at approximately the same time, the simplex method was widely used in economics and made possible for economists to formulate large models and analyze such models in more systematic ways.

Another class of linear optimization methods was developed in succession to simplex methods. This class is known as *Interior-Point* optimization methods

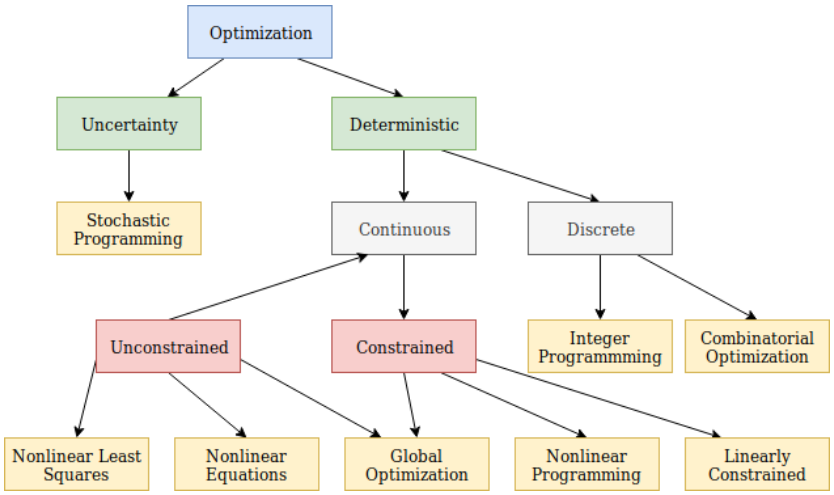


Figure 2.1: The taxonomy of optimization methods

and they are proved to be a strong competitor to simplex methods on large-scale problems. First interior-point method was called *ellipsoid method* and was proposed by Krachiyan [4]. This method was found to be inefficient in comparison to the simplex method because for all problems it approaches its worst case of the polynomial. But the new direction of the linear optimization, became interesting and soon new methods arrived. In 1984, Karmarkar [5] showed his *projective algorithm* that was still had polynomial complexity, but the work itself provided the great mathematical framework to work upon.

One of the most noticeable differences between the simplex method and interior-point methods is that iteration of interior-point methods is very expensive to compute but makes significant progress towards the solution. On the other hand, the simplex methods have a lot of inexpensive iterations until reaching satisfactory results.

2.1.3 Nonlinear Optimization

Nonlinear optimization is wildly used in natural sciences and engineering, especially in the field of computer vision. There are a great amount of problems that could be modeled into the optimization task of nonlinear form. Thus, before we will be discussing the application of nonlinear optimization methods we should cover the theory behind it.

Nocedal and Wright [3] describe two classes of existing nonlinear optimization methods that are conceptually complementing to each other. *Line Search* methods first try to find a descent direction along which the objective function will be reduced and then computes a step size that decides how far should move along that direction. The descent direction can be computed by various methods, such as gradient descent, Newtons method, and Quasi-Newton

method. The step size can be determined either exactly or inexactly. *Trust-Region* methods approximate the objective function using a model function (often a quadratic) over a subset of the search space known as the trust region. If the model function succeeds in minimizing the true objective function the trust region is expanded; conversely, otherwise it is contracted and the model optimization problem is solved again.

Another method to solve nonlinear optimization problems was proposed in the 1960s by Fletcher and Reeves. This method is called nonlinear conjugate gradient method and is the earliest known technique for solving large-scale nonlinear optimization problems.

Least-squares problems are present in many areas of applications and they are the largest source of unconstrained optimization problems. In the least-squares problem the objective function has the following form:

$$f(x) = \frac{1}{2}\sum_{j=1}^{m} r_j^2(x), \tag{2.3}$$

where $r_j : \mathbb{R}^2 \implies \mathbb{R}$ and usually referred to as a *residual*. A lot of optimization problems in physics, computer vision, engineering are modeled using least-squares methods. The advantage of such methods is that such special form of the objective function f makes it easier to solve than general unconstrained optimization problems.

One of the most popular algorithms for nonlinear least squares problems is *Gauss-Newton* method. This method can be viewed as modified Newton's method with *line search*. Another algorithm that is basically the same as Gauss-Newton except it uses *trust-region* strategy is *Levenberg-Marquardt* method. For large-residual problems, the convergence rate of Gauss-Newton and Levenberg-Marquardt algorithms is slower than the general unconstrained problems, such as Newton or quasi-Newton. However, their early iterations (before reaching a neighborhood of the solution) could be much slower than in Gauss-Newton and Levenberg-Marquardt. Thus, there is an adequate idea to construct hybrid algorithms that incorporate the advantages of both ways. Roger Fletcher [6] proposed one approach to solve efficiently large-residual problems. This approach incorporates the best of the general unconstrained methods and least-squares methods.

In contrast to the Levenberg-Marquardt method, another trust-region method that could be used for solving nonlinear least squares problem is *Dog-leg* method that was firstly introduced by M.J.D. Powell. The key advantage of the Dogleg over Levenberg-Marquardt is that if the step computation for a particular choice of does not result in sufficient decrease in the value of the objective function, Levenberg-Marquardt solves the linear approximation from scratch with a smaller value of μ. Dogleg on the other hand, only needs to compute the interpolation between the Gauss-Newton and the Cauchy vectors, as neither of them depends on the value of μ.

Another class of problems in nonlinear optimization concerns not with optimization some objective function, but with solving a particular set of nonlinear

equations. Mathematically, the problem could be written as

$$r(x) = 0, \tag{2.4}$$

where $r : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ is a vector function of form

$$r(x) = \begin{vmatrix} r_1(x) \\ r_2(x) \\ \vdots \\ r_n(x) \end{vmatrix}$$

Particular methods for solving nonlinear equations include themselves *Newton's* and *Broyden's* methods. Practical methods that solve a particular problem are based again on *line search* and *text-region* approaches; such methods improve global convergence of the solution. Other methods impose very close connection to the nonlinear least-squares problems.

Solving nonlinear optimization problems algorithms such as *Dense QR factorization*, *Dense Normal Cholesky Decomposition*, *Sparse Normal Cholesky Decomposition*, *Dense Schur's complement*, *Sparse Schur's complement*, *Iterative Schur*. It will be seen later that a lot of implementations of nonlinear optimization methods heavily rely on these algorithms.

2.2 Optimization Frameworks

As it was stated, there are a lot of different methods for solving optimization problems. All of these problems should be modeled beforehand and then executed on the computational machine. That is why the need for efficient optimization framework arise; it should give the opportunity to model the problem and it should provide with the solvers that could process this model and find a feasible solution. One of the most popular and wildly used optimization frameworks today are *Ceres solver* and *g2o*.

2.2.1 Ceres Solver

Ceres solver is nonlinear optimization framework that is developed and maintained by Google. It is written in C++ as almost all solvers, because it impose high speed calculations without so much overhead. Currently, Ceres solver can solve (1) nonlinear least squares problems with bounds constraints and (2) general unconstrained optimization problems.

For solving nonlinear least squares problems, Ceres implements both Trust-region methods and *line-search* methods. *Trust-region* methods include popular Levenberg-Marquardt (with exact step [7] and inexact step [8]) and Dogleg methods. *Line search* implemented methods are could be used only for unconstrained problems. Such methods include *Steepest Descent*, *nonlinear conjugate descent*, *BFGS*, and *LBFGS*.

There are two Dogleg implementations. *Traditional Dogleg* as described by Powell constructs two line segments using the Gauss-Newton and Cauchy vectors and finds the point farthest along this line shaped like a dogleg (hence the name) that is contained in the trust-region. *Subspace Dogleg* is a more sophisticated method that considers the entire two-dimensional subspace spanned by these two vectors and finds the point that minimizes the trust region problem in this subspace.

Ceres solver provides a convenient API for constructing an optimization problem. At the same time, Ceres solver provides API that has methods for solving the constructed problem.

2.2.2 g2o: Graph Optimization Framework

A general framework for graph optimization (g2o) allows performing optimization of least squares problems that can be embedded as a graph or in a hyper-graph. It is a popular framework that is currently used in many SLAM systems [9, 10] Giorgio Grisetti et al. presented their new g2o optimization framework for solving nonlinear least squares problem in graph or hyper-graph space. They state that Such a way of solving the problem is convenient whilst trying to optimize SLAM solutions or other robot systems. g2o implements two algorithms for solving nonlinear least squares optimization problems: Levenberg-Marquardt and Gauss-Newton as well as Powell's dogleg. In Figure 2.2 we can see the class diagram of g2o optimization framework that shows the internal structure of the library. t Luca Carlone held detailed study on the comparison of two optimiza-

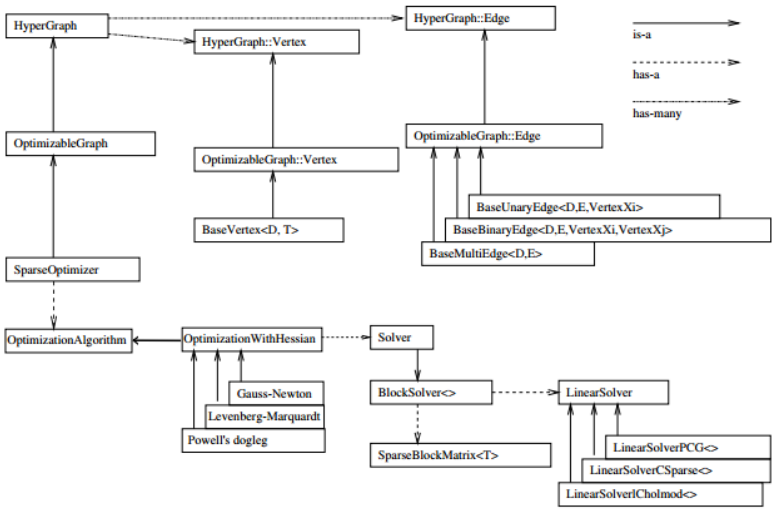


Figure 2.2: Class Diagram of g^2o [9]

tion frameworks g2o and toro. He stated that both frameworks use different

cost function and in order to assess their difference objectively one needs to take into the account.

2.3 Simultaneous Localization and Mapping

2.3.1 Problem Definition

According to Durrant-Whyte and Bailey [11], **simultaneous localization and mapping** is the method of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within the built map. Thus, solving the SLAM problem allows mobile robot with sensors like camera or laser to build a map of the environment as well as determine its current location.

Arun Das [12] from the University of Waterloo showed a great visual representation of SLAM overview. The figure 2.3 shows the SLAM system as an aggregation of two layers: front-end and back-end. *Front-end* layer is responsible for feature extraction, sensor, and measurement handling. *Back-end* solves optimization tasks that are defined in the SLAM problem, e.g. minimization of reprojection error, minimization of pose error, etc. Most of the optimization problems in SLAM are formulated as nonlinear optimization problems.

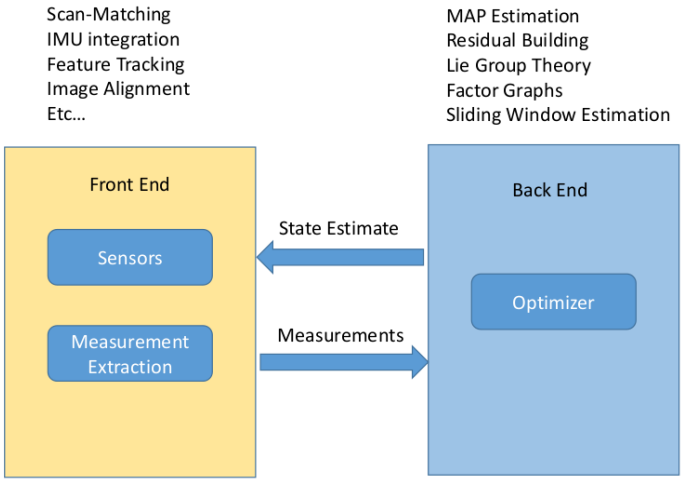


Figure 2.3: Typical SLAM system overview [12]

2.3.2 Types of SLAM

There are a lot of different types of solutions for SLAM problems that are needed to be outlined. Jakob Engel et al. [13] present their taxonomy of currently popular types of methods that solve SLAM and Visual Odometry problems. They

stated that methods to solve the SLAM problem can have two representative properties:

1. **Indirect or Direct methods**
    - Indirect (Feature-based) methods consist of two steps. Firstly they pre-process raw sensor measurements in order to create an intermediate representation, also simultaneously computing image coordinates for corresponding points. Then they use values of computed representation in order to estimate geometry and camera motion.
    - Direct method skips the first step of pre-processing raw data and directly uses values from the sensor(in our case it is the light received from a particular direction at a particular time).
2. **Dense or Sparse methods**
    - Dense methods attempt to use and reconstruct all pixels in the 2D image domain. The fundamental difference between dense and sparse methods is related to the addition of geometry prior. The sparse formulation does not give us the notion of neighborhood and geometry parameters are conditionally independent given the camera poses. On the other hand, dense formulation exploits neighborhoods in order to formulate geometry prior.
    - Sparse methods use and reconstruct only selected set of independent points(corners).

Thus, according to the above definitions, they derived four classes of methods for solving SLAM problems, that will be briefly described below.

- **Sparse + Indirect**: generate 3D geometry out of a set of keypoint matches. Examples of such works: monoSLAM [14], PTAM [15], ORB-SLAM [1]
- **Dense + Indirect**: estimate 3D geometry from dense optical flow field. Example: [16]
- **Dense + Direct**: these methods use photometric error and geometric prior to estimate geometry. Examples are DTAM [17], REMODE [18] and LSD-SLAM [19].
- **Sparse + Direct**: these methods incorporate photometric error but without using geometric prior. Examples: DSO [13]

BRISK: Binary Robust Invariant Scalable Keypoints is a feature extractor system developed by Stefan Leutenegger et al. for Autonomous Systems lab in ETH Zurich. BRISK defines a novel approach for feature extraction: firstly algorithm performs **scale-space keypoint detection** and them performs **keypoint description**.

2.3.3 Important SLAM Implementations

Now it is needed to make an overview of the most common SLAM solutions that are present today.

ORB-SLAM

In 2015, Raul Mur-Artal et al. [1] presented their newly designed feature-based monocular SLAM system. It implements such features like tracking, mapping, relocalization and loop closing.

Figure 2.4 describes the overall architecture of ORB-SLAM. ORB-SLAM's

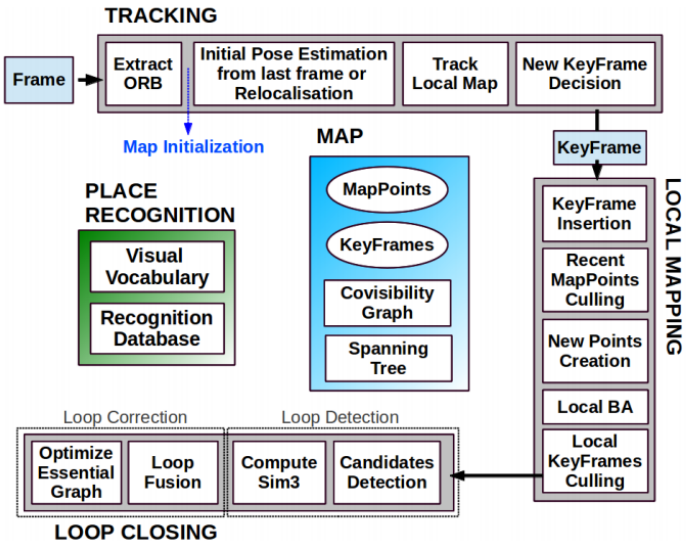


Figure 2.4: Architecture of ORB-SLAM1

front-end has ORB feature extractor that is much faster than SIFT, SURF and A-KAZE feature extractors. The back-end consists of three threads that are run in parallel: tracking, local mapping and loop closing. *Tracking thread* localizes the camera with every frame and decides when to insert a new keyframe. *The local mapping thread* processes new keyframes and performs local bundle adjustment to achieve an optimal reconstruction in the surroundings of the camera pose. Local mapping is also in charge of culling redundant keyframes. *Loop closing thread* searches for loops with every new keyframe. If a loop is detected, the similarity transformations should be computed in order to inform about the drift accumulated in the loop. Then both sides of the loop aligned and duplicated points are aligned. After that pose graph optimization over similarity constraints is performed to achieve global consistency.

ORB-SLAM uses the Levenberg-Marquardt nonlinear least squares method implemented in g2o in order to carry out all optimization problems.

LSD-SLAM

LSD-SLAM (2014) is the novel approach to SLAM problem designed by Jakob Engel et al. [19] LSD-SLAM is a direct (feature-less) monocular SLAM algorithm that allows building large-scale, consistent maps of the environment. Along with highly accurate pose estimation based on direct image alignment, the 3D environment is reconstructed in real-time as pose-graph keyframes with associated semi-dense depth maps. The novelties include: (1) a novel direct tracking method which operates on the sim(3), thereby explicitly detecting scale-drift; (2) an elegant probabilistic solution to include the effort of noisy depth values into tracking. One of the optimization minimization problems that LSD-SLAM solves is the minimization of photometric error between two images; it solved by the Gauss-Newton optimization method.

The SLAM algorithm itself consists of three major parts: tracking, depth map estimation, and map optimization. *Tracking* component continuously tracks new camera images and estimates their rigid body pose with respect to the current keyframe. *Depth map estimation* uses tracked frames to either refine or replace the current keyframe. Once a keyframe is replaced as tracking reference it is incorporated into the global map by *Map optimization* component.

DSO

Jakob Engel et al. [13] proposed a novel direct sparse visual odometry formulation. It combines a fully direct probabilistic model (minimizing a photometric error) with consistent, joint optimization of all model parameters, including geometry - represented as inverse depth in a reference frame - and camera motion. They incorporate the minimization of photometric error from the Leutenegger et al. [20] in a sliding window using the Gauss-Newton algorithm, which gives a good trade-off between speed and flexibility. Also, they incorporate marginalization step, when they drop old variables using the Schur complement.

The front-end part of DSO performs data-selection and provides accurate initialization for optimizing the highly non-convex energy function.

ICE-BA

Not long ago, the new study on visual-inertial SLAM systems was provided by Haoimin Liu et al. They have developed and implemented new SLAM system which they called Incremental, Consistent and Efficient Bundle Adjustment (ICE-BA). It is a fully fledged SLAM system for which they've written their own optimization solver in order to optimize nonlinear cost functions. **Bundle Adjustment** [21] - the problem of simultaneously improving coordinates of the scene, parameters of relative motion, and optical characteristics of the camera(s) employed to acquire the images, according to an optimality criterion involving the corresponding image projections of all points. Existing solutions include **ICE-BA: Incremental, Consistent and Efficient Bundle Adjustment for Visual-Inertial SLAM** [22], that leverages the sparseness and the unique

matrix structure for the optimization of sliding window based bundle adjustment and propose new marginalization strategy to improve global consistency.

2.3.4 Visual-Inertial Fusion

It's a popular idea to integrate visual and inertial measurements together. In this case, a scene captured by the camera, together with estimates from gyroscopes and accelerometers. First ideas of visual-inertial fusion for pose estimation are associated with filtering [23, 24].

The inertial measurement unit (IMU) is a device that performs the measurement of different forces (motion, magnetic), angular position, etc. Known examples of IMU's are an accelerometer, gyroscope, magnetometer, etc. The accelerometer measures physical acceleration whereas gyroscope measures angular velocity and body's orientation. All of these IMU's that were described above are wildly used in robotics, especially in developing systems for unmanned aerial vehicles (UAV) and for mobile phones. As the computational power of mobile phones increases each day, they become more used in the application of robotics, specifically in solving SLAM problems. Today mobile phones have few IMU's that can be used in solving particular robotics tasks.

Stefan Leutenegger et al. [20] state that visual-inertial fusion methods can be categorized into two classes: *loosely-coupled* and *tightly-coupled*.

- The *Loosely-coupled* approach considers the IMU measurements to be processed independently into vision optimization.
- The *Tightly-coupled* approach tries to jointly fuse all sensors states.

2.3.5 SLAM Optimization Tasks

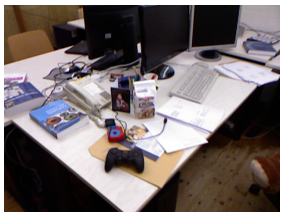
As it was described above, each SLAM implementation has its own design model and they consist of different optimization tasks. But all of them somewhat have optimization tasks in common. Two of the most important optimization tasks in feature-based SLAM systems is *local bundle adjustment* and *global bundle adjustment*. Bundle adjustment concerns about the minimization of reprojection error and can be modeled as nonlinear least squares optimization problem. Direct-based SLAM systems that do not extract features from the images, but instead use the whole image instead, usually define optimization task in terms of minimizing the photometric error based on image pixel intensities. Another task is Pose graph optimization which is concerns with refining the pose.

2.4 SLAM Datasets

As it was discussed earlier, the main idea of SLAM algorithms is to build the initial map (point cloud), then gather data from sensors (images from cameras, other measurements from IMU, etc.), then localize robot against the map and then refine (optimize) the map accordingly. Sometimes researches only look

for testing their SLAM algorithms outside the robot environment; it is much simpler and convenient to test their prototype algorithms on already existing datasets. Thus, researchers from all over the world create various datasets that could contain (1) image sequences for mono/stereo camera, (2) readings and measurements from IMU's, (3) ground truth trajectory to assess the performance of SLAM algorithm, (4) calibration files. The main goal for creating such datasets for evaluation of SLAM algorithms is to provide researches from all over the world with the means of assessing each other's algorithms. This helps researchers to compare different SLAM algorithms based on performance on a specific dataset. Thus, this section will describe the most popular and widely used datasets that are used for validating and assessing the performance.

Computer vision group of Technical University of Munich (TUM) informatics department provides a great amount of datasets for monocular SLAM, RGB-D SLAM, visual-inertial SLAM, etc [25,26]. For example, datasets for RGB-D contain a sequence of RGB images, depth data that was gathered from the camera, ground-truth trajectory and accelerometer data [26]. A project of Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago (KITTI) provides a lot of datasets that were captured by the camera mounted on a vehicle[27, 28]. Another big set of datasets gathered by M. Burri et al. [29] for Europe Robotics Challenges (EuRoC). These datasets contain data gathered from Micro Aerial Vehicle (MAV) and includes data from the machine hall, etc. Figure 5.4 shows samples of data from TUM, KITTI, and EuRoC datasets; these are camera observations that would be later fed to SLAM algorithm alongside with other supplementary information.



(a) TUM: RGB-D freiburg1 dataset of indoor desktop sequence

(b) TUM: RGB-D freiburg3 dataset of indoor office sequence

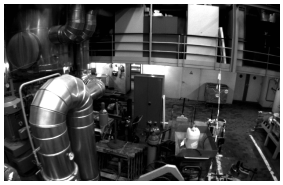
(c) TUM: Monocular image sequence

(d) KITTI: Vehicle outdoor dataset

(e) KITTI: Vehicle outdoor dataset

(f) KITTI: Vehicle outdoor dataset

(g) EuRoC: Machine hall (MH1) first dataset



(h) EuRoC: Machine hall (MH2) first dataset

(i) EuRoC: Vicon room 1

Figure 2.5: TUM, KITTI, and EuRoC datasets

Chapter 3

Methodology

This chapter will emphasize on the system design issues and the implementation of program for comparison different nonlinear optimization techniques. Generally, there are three main ways of how to compare nonlinear optimization methods in the context of SLAM algorithms:

- **Matlab model** - matlab is very powerful tool for building fast and efficient prototype models. So the first choice for comparing nonlinear optimization technique would be implementing a SLAM prototype model, designing optimization problems; After the model is built, optimization problems could be modified in order to assess overall performance of the matlab model. Particularly, matlab provides Optimization toolbox, which is a big library with great amount of methods; each method could be tweaked according to the needs of the researcher who wants to compare various methods.

- **Implement SLAM system from scratch** - this choice demands the researcher to develop full SLAM system; here the problem of choosing hardware and sufficient sensors is of great deal. The algorithm should be fast and efficient, so design of such algorithm would take a lot of considerations like programming language (usually C++ is the choice), linear algebra libraries (some of them are implemented in Fortran), nonlinear optimization library, etc. One would argue that the most difficult parts of SLAM system could be assembled from existing open-source libraries; it would ease the development process - you need to carefully "glue" components together. When the SLAM algorithm will be implemented, nonlinear optimization problems could be tweaked and compared; you might need to re-implement optimization problems if you want to compare different methods from several different optimization frameworks.

- **Change existing SLAM implementations** - the key idea is to use already implemented SLAM system and alter its code base in order to perform comparison analysis of nonlinear optimization methods. It is also

convenient way of proceeding the research, because the performance of altered SLAM could be compared to existing results of researchers who implemented the algorithm.

The final approach for comparing nonlinear optimization methods that was chosen is the approach of changing existing SLAM system. For successful completion of such problem it was decided to (1) choose good existing SLAM library, (2) analyse its default nonlinear optimization methods, (3) choose and implement new nonlinear optimization methods using another optimization framework, (4) develop a simple application with GUI that would give an opportunity to easily set up the configuration of SLAM's optimization methods, (5) save metrics that assess the performance of SLAM algorithm in some database and (6) provide the mean to visualize metrics, errors for particular configurations. The design approaches of these problems will be described with the great extent later in this chapter.

3.1 Choosing SLAM system

3.1.1 SLAM system

As it was previously stated in Chapter 2, it is needed to choose front-end that extracts and processes features out of images and back-end that refines reprojection errors and solves other optimization problems like global or local bundle adjustment and etc. There are a lot of various SLAM systems, feature extractors and optimization problem solvers that we can choose from. For this thesis ORB-SLAM2 library [2] was chosen as the base SLAM system. It has its own ORB feature extractor that serves itself as a front-end part of the SLAM system; g2o graph optimization framework is used as a part of a back-end that solves optimization problems.

Motivation behind choosing ORB-SLAM2 is that it is fairly new SLAM system, it works really good with indoor settings; also it is computationally efficient due to feature extracting and tracking fixed amount of points. Thus it could be used on mobile phones without much of a performance pain.

3.2 ORB-SLAM

3.2.1 System Overview

In this section we will describe initial framework for testing different nonlinear optimization methods that can be used in order to increase performance of SLAM system. For SLAM system, we've chosen ORB-SLAM that provides two layers of the system:

- **Front-end: ORB Extraction and Matching** - this layer of SLAM system performs feature extraction and matching in camera frame.

- **Back-end: g2o optimization framework** - this layer uses g2o graph optimization framework in order to solve optimization problems.

3.2.2 Nonlinear Optimization

Using ORB-SLAM as the starting point, it is needed to be identified what part of the system should be altered in order to perform comparison of nonlinear optimization techniques. In order to perform comparison we need to change code in the back-end of the system, particularly in the optimization process. Because of the fact that ORB-SLAM uses g2o framework to solve optimization tasks, the main work should be involved with working with altering configuration of g2o solvers and other functions. G2o optimization framework implements linear and non-linear solvers that could be used as a working method for performing optimization process. Particularly, g2o implements these non-linear optimization methods: Levenberg-Marquardt, Powell's Dogleg and Gauss-Newton methods.

Backend of ORB-SLAM solves number of optimization problems. We are interested in non-linear optimization problems that are present in there. There are five non-linear optimization problems that are solved during SLAM tracking: global bundle adjustment (GBA), local bundle adjustment (LBA), Pose Graph Optimization over SO(3) Constraints, Relative Sim(3) Optimization and Essential Graph Optimization. By default, ORB-SLAM solves these non-linear optimization problems using Levenberg-Marquardt method. So that means in order to test other optimization methods we need to call different g2o functions that are responsible for corresponding optimization method.

3.3 Extending ORB-SLAM Optimization

This section will describe nonlinear optimization techniques that were chosen for comparison between each other. We will describe each techniques more closely than in chapter 2.

As a part of the thesis, it was decided to extend the functionality of ORB-SLAM library. The motivation behind this decision is that g2o optimization framework provides limited amount of methods for solving non-linear optimization problems. Ceres solver was decided to be used as second way to perform optimization tasks in ORB-SLAM. It has more flexible code base, that allows fine-tuning of optimization models and it implements more optimization methods that also can be easily tuned. Also, we need to extend ORB-SLAM in such a way so that performance of ORB-SLAM with new optimization methods could be easily assessed and evaluated.

3.3.1 Choosing Nonlinear Optimization Methods

In Chapter 2, Ceres solver was described as the optimization framework that concerns with solving constrained and unconstrained non-linear optimization

problems, e.g. non-linear least squares problem. In ORB-SLAM's default implementation all 5 optimization problems are modelled as non-linear least squares problem.

As it was discussed in previous chapter, ceres solver has 2 types of methods for solving nonlinear optimization problems: *Trust-region* and *Line Search*. *Trust-region* methods include itself such methods as Levenerg-Marquardt and Dogleg. On the other hand, *Line search* methods include itself Steepest descent, nonlinear conjugate descent, BFGS and LBFGS methods. All of those methods are flexible in terms of configuration; one could easily adjust chosen algorithm to their needs using C programming language Options structure.

The advantage of these implementations over g2o's is that Ceres solver gives an opportunity to adjust how exactly Levenberg-Marquardt or Dogleg methods will be solved; there are several types of each of these methods that could influence the solution differently like Levenberg-Marquardt with inexact/exact step, Powell's traditional Dogleg or Subspace Dogleg. There are a lot more small configurable variables that could impact on the performance of the optimization process, e.g. radius of region, number of iterations and others hyperparameters that could be adjusted.

Ceres also give a rise to Line Search non-linear optimization methods that could be used for optimizing unconstrained problems (like in ORB-SLAM). We will chose such line search methods:

- *Steepest Descent Line search* - this corresponds to choosing $H(x)$ to be the identity matrix. This is not a good search direction for anything but the simplest of the problems. It is only included here for completeness.
- *Nonlinear conjugate descent* - is a generalization of the Conjugate Gradient method to non-linear functions. The generalization can be performed in a number of different ways, resulting in a variety of search directions. Ceres Solver currently supports Fletcher-Reeves, Pola-Ribiere and Hestens-Stiefel directions.
- *BFGS* - is a generalization of the Secant method to multiple dimensions in which a full, dense approximation to the inverse Hessian is maintained and used to compute a quasi-Newton step. BFGS is currently the best known general quasi-Newton algorithm.
- *LBFGS* - it is limited memory approximation to the full BFGS method in which the last M iterations are used to approximate the inverse Hessian used to compute a quasi-Newton step

3.3.2 Designing Optimization Extension

As it could be seen in the figure 3.1, there is an ORB-SLAM implementation that has default g2o optimization framework binding. Extension to such default architecture is the inclusion of Ceres solver to ORB-SLAM environment. Next, for each optimization task that is defined for ORB-SLAM the corresponding

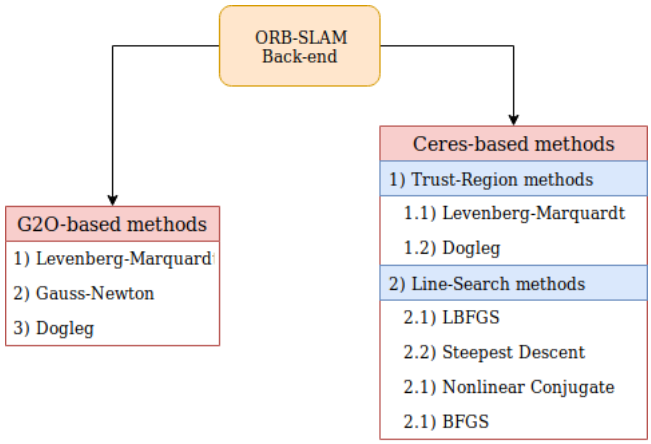


Figure 3.1: Scheme of ORB-SLAM back-end optimization methods after extension

model written using Ceres solver. Later the parameters of optimization problem solvers will be easily adjusted in Comparison application, that would be also developed as a part of the thesis.

After the implementation of these additional ways to perform optimization tasks in ORB-SLAM, Comparison application is designed that would give the user an opportunity to set needed parameters for evaluation of ORB-SLAM's performance with these parameters.

3.4 Comparison Application Design

This section is describes one of the main goals of this thesis - design and implement Comparison application that would allow one to perform an assessment of influence of non-linear optimization methods on ORB-SLAM's performance.

First of all, the performance measure of the non-linear optimization problem needs to be defined. Actually, there can be several performance measures in ORB-SLAM. The most obvious one is the time of the execution of the optimization task. It is quite easy to assess the time of execution of each optimization task by simply putting a timer in the code inside corresponding places where the optimization function is called from. Another performance measure that can be chosen is the overall error between ground truth trajectory of the dataset and the reconstructed trajectory. Yet another important measure of performance is CPU consumption; different optimization methods take different CPU resources in order to solve corresponding tasks. That is the reason for finding the ideal methods that would be best for our problems.

The motivation for designing and developing such a framework is concerned with the nature of comparing different non-linear optimization methods in ORB-SLAM. There are 5 optimization problems that are defined to be solved in

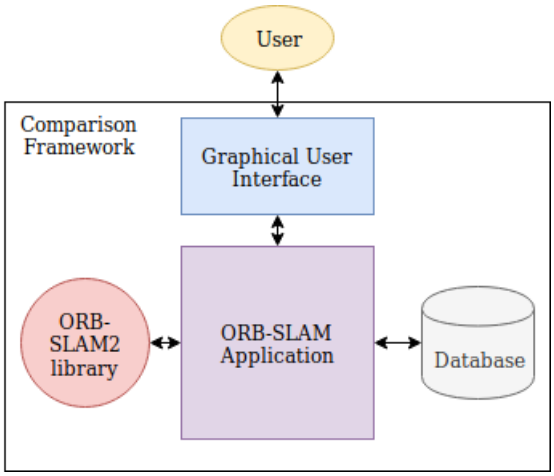


Figure 3.2: High-level prototype design of Comparison Framework

ORB-SLAM. Given the vast amount of optimization methods for each of these optimization problems the problem arises - there are quite many combinations of different optimization methods used together. If one would compile ORB-SLAM each time with different optimization methods set, then it would take a long time to waste. That is why the idea of generalizing ORB-SLAM library for different nonlinear optimization methods and creating a GUI application to set the configuration for the SLAM system is really interesting.

In the figure 3.2 the prototype design of the comparison framework is proposed. The architecture of such application will be discussed from bottom to top:

1. **GUI application** - this part of the desktop application gives an opportunity for user to set needed parameters like what optimization method will be used in particular optimization task; what dataset will be used for assessing current ORB-SLAM's configuration; number of features, etc; When user will finally decide on all parameters to be set, then the experiment will be visualized in place. After the experiment one can access the results of previous experiments and may visualize evaluation metrics using visualization tools.

2. **ORB-SLAM Example** - it is core application that uses ORB-SLAM2 library in order to process image sequence (dataset) to produce the point cloud map and trajectory. Default examples are created for each type of dataset when building the ORB-SLAM2 library from source. Ideally, there is the need of implementing one general application that would take different datasets.

3. **ORB-SLAM2 Library** - this is the core of orb-slam that implements ORB feature extractor and the rest of back-end. This thesis defines that

ORB-SLAM back-end will be consisted of g2o based optimization methods and Ceres-based optimization methods. Each of the newly defined method should be controlled from the outside, particularly from higher level.

4. **Database** - this is the storage that would contain all the results from the ORB-SLAM experiments. Later those results could be further visualised in GUI of the application. Figure 3.5 shows prototype database tables that will store the data from running experiments.

The end result should represent the user application that would give an opportunity for the user to test different ORB-SLAM configurations; after setting the experiments (parameters, etc.) they can be ran and results should be stored for the research purpose.

3.4.1 GUI Application

For the implementation of GUI application, Qt5 QWidgets were used that allow to design and create scalable GUI applications. The idea behind this GUI application is that the user can specify exactly how nonlinear optimization task will be solved. The main configuration concerns with the method for solving such problems; the user also can specify advanced settings that allow fine-tuning the algorithm's work. Advanced configurations will differ for each optimization framework g2o and ceres.

Figure 3.3 presents the start up a window that gives the user an opportunity to launch the ORB-SLAM with a specific configuration. For each optimization problem, user can pick the library in which those Optimization problems will be formulated, optimization methods, advanced configuration for Ceres-based problems and dataset on which the ORB-SLAM will perform the localization and mapping.

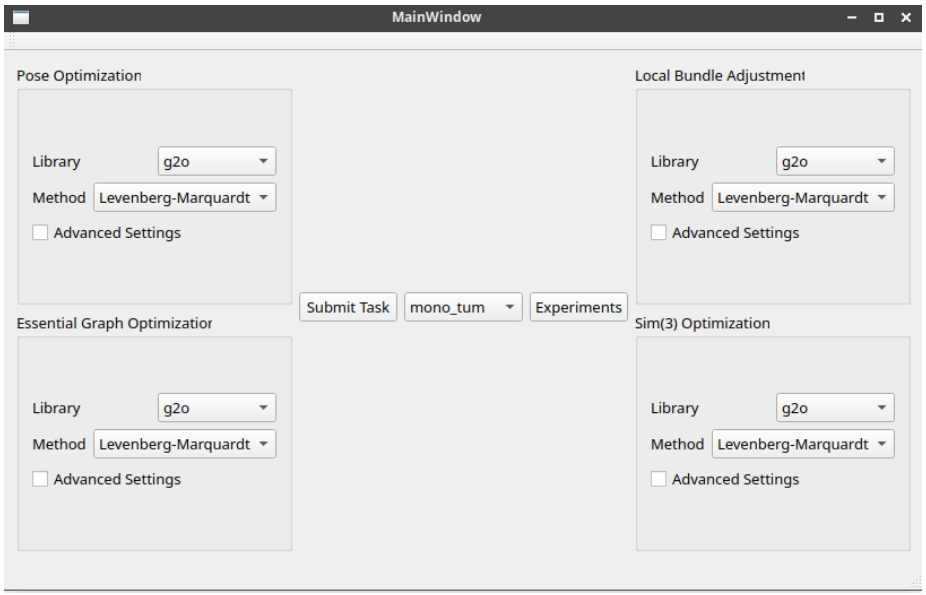


Figure 3.3: Initial GUI page that allows to configure and launch specific experiment

The underlying idea is that after the user presses the button "Submit Task" application will generation JSON file from the form. Then ORB-SLAM application will be started which will initialize the SLAM system using JSON configuration file. After ORB-SLAM finishes its work, the GUI application fills the database with the information about finished experiment; it will store all results that are needed for visualization. The button "Experiments" opens a new window that is used for visualization of the experiments and for checking the results of ORB-SLAM performance with a specific configuration.

Figure 3.4 shows second GUI page that give the user several possibilities:

1. **Look up results** from past experiments. The functionality allows to also check experiments and visualize them
2. **Trajectory visualization** this button launches the visualizing tool that will show the trajectories of chosen experiments.
3. **APE visualization** - absolute pose error shows the derivation of generated trajectory from the reference.
4. **Unified statistics** button will gather several saved results from APE evaluation in one place.

All visualizations are handled by *evo* [30] that is a library for visualizing odometry and SLAM written in Python. There are 3 important applications that are provided by Evo library:

- *evo_traj* - tool for visualizing trajectories. It also can be used for comparing the reference trajectory with the generated one.

- *evo_ape* - tool for evaluating the generated trajectory against the reference trajectory based on specific metric: Absolute pose error.

- *evo_res* - unifies several results of metrics calculation and produces nice plots for comparing the trajectories.

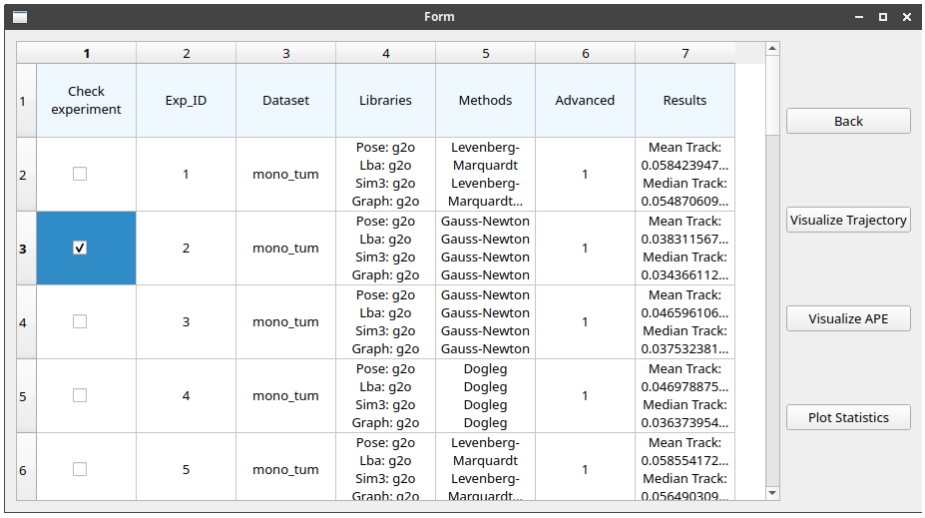


Figure 3.4: Second GUI page that allows user to see the results of past experiments and visualize evaluation

3.4.2 ORB-SLAM Application

Default ORB-SLAM example applications are built together with the library during the compilation process. Each example is related to a particular dataset and particular sensor settings (whether the camera is stereo, mono or RGB-d). First of all, each of these examples initializes SLAM system: load configuration files, calibration files, camera mode flags; the SLAM system initialization should be extended in order to process JSON file with additional configurations for SLAM optimization tasks. During initialization SLAM system launches 3 additional threads: one for loop closing, second for LBA, third for visualization of the point cloud, camera trajectory and tracked points layered over an initial image sequence. The main thread of the SLAM system for each image performs tracking. Each of these examples implements little functions that help to process the corresponding datasets into SLAM tracking function.

3.4.3 ORB-SLAM2 Library

ORB-SLAM library contains functionality that makes the SLAM algorithm work. To make the application work it is needed to extend some of the functionality, e.g. ORB-SLAM Optimizer should be extended with new implementations for optimization tasks using Ceres solver library, add functionality that can parse JSON configuration file and perform optimization accordingly to this JSON configuration file.

3.4.4 Database

In order to perform an assessment of different configurations, some kind of database that would store the results of the experiment should be designed and implemented. It was decided to design relational tables that would encompass metrics that would be compared between each other later. Simple relational database design presented in Figure 3.5 shows the way the data will be saves upon execution of the experiment (running SLAM algorithm with the new configuration on the particular dataset). Later this data could be used for visualizing in the GUI application. The high-level hierarchy emphasizes on what data we want to store; each entity represents a future table. *Experiments* table is the main table that is mainly composed of reference key on other databases; it should contain the information about what dataset we are using, what set of optimization methods were chosen, what libraries will be used for solving optimization problems, store the results (median tracking time, mean tracking time, absolute pose error between reference trajectory and generated trajectory). Also, it is worth to mention that *Experiments* table is constrained with reference (foreign) keys on other tables, which ensures that the insertion of data to the experiment table will be the last operation. In the Implementation chapter 4 the final specific database design will be presented.

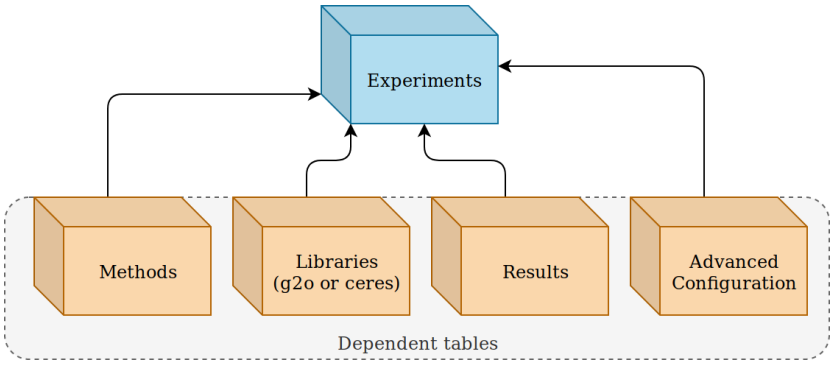


Figure 3.5: High-level scheme for database that will store the results of experiments

3.5 Choosing Datasets

All of the SLAM researchers that were refered in chapter 2 based their evaluation of their algorithms on the datasets. There are some credible sources that provide great datasets that include: image sequence, ground truth trajectory, etc. For the purpose of this thesis, the different datasets should be chosen for further usage in ORB-SLAM evaluation inside Comparison application.

The datasets were chosen based on the fact that they were initially supported by ORB-SLAM default examples. Each of these examples have a little function implemented that takes corresponding dataset and puts it in the array. So the final list of datasets that were used during this research is: *TUM*, *KITTI*, and *EuRoC* datasets. Each of them provide corresponding calibration file with camera parameters that are fed to the SLAM algorithm, image sequences for different kinds of cameras, e.g. monocular, stereo, RGB-D, etc. These datasets will be stored on the computer and the path to these datasets will be passed as an argument to the ORB-SLAM examples through GUI application.

3.6 Designing Experiments

After implementing Comparison application, the reason arises on how exactly to organize the performance assessment of different configurations of ORB-SLAM. The application allows the user to launch one task or several in a row. So before running the experiments to gather results we want to carefully design batches of configurations to be tested. It will give an opportunity later to group and compare results based on some meaningful reason; for example, comparing the performance between optimization tasks implemented using g2o library and ceres solver library. This section will describe experiments that could possibly bring interesting results that show the correlation between the nonlinear optimization method being used in a particular optimization problem and the overall performance of the SLAM system interpreted in terms of meaningful metrics that were discussed earlier.

The batches of experiments could be created in a hierarchical way. Figure 3.6 shows the overall hierarchical structure of the experiments. For example to extract one instance of experiment one would like to go from top level to bottom level; first, you need to choose dataset which will be fetched through the SLAM algorithm; second, for each optimization problem (there are only 4 that are under consideration) choose optimization framework (g2o or ceres) through which they will be executed; third, for some optimization problems choose the advanced configuration for optimization frameworks. So the hierarchy just shows the general approach to how to choose one experiment. Of course, one can somewhat automatize the process of constructing the experiment.

Because there are a lot of combinations of experiments due to a high number of variables that could be permuted only specific batches of experiments could be formed. Formation of such batches of experiments, as it was already mentioned above, can be performed by thinking of what experiments are of the

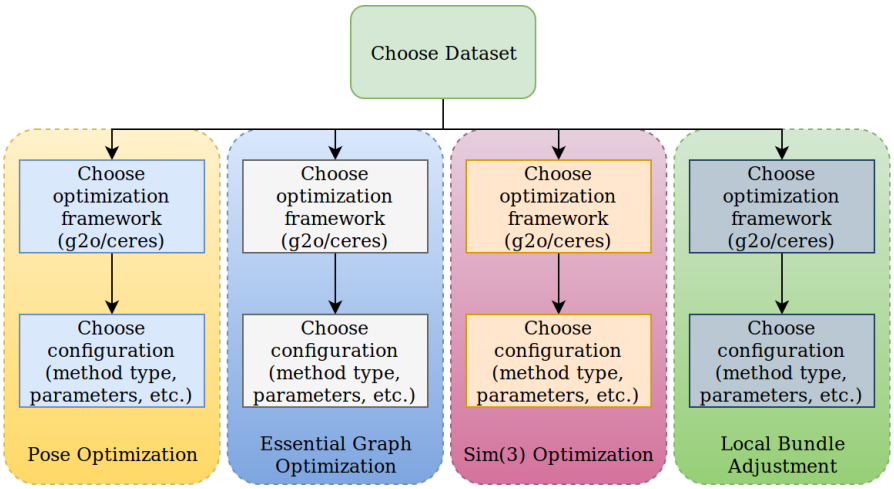


Figure 3.6: Hierarchy of experiments

interest for the research. Below is the list of such interesting questions that will help to form batches of experiments; results of these experiments can later answer on these questions. Also in some of the experiments, we want to have some reference, baseline which will be compared to the results of the experiment. In the case of ORB-SLAM2 library, the reference will be the initial configuration of optimizer; as it was already stated, the initial configuration is for every optimization problem use g2o optimization framework with Levenberg-Marquardt method set for solving nonlinear optimization tasks.

1. For each dataset what overall library performs better in terms of specific metric (tracking time, trajectory error, etc)?

2. For one of the four optimization problems, what optimization method performs better in the scope of the fixed library? (Example: For Pose Optimization problem, what is the best optimization method Levenberg-Marquardt or Dogleg if optimizer uses only Ceres-based optimization problems?)

3. How much fine-tuning of advanced configuration for optimization methods impacts the performance of the algorithm?

These general question can lead to specific experiments that could be ran using the Comparison application's GUI. Results of the experiments will be saved in the database which later can be retrieved and visualized; these results and plots that can show the correlation between different configurations and the overall performance of ORB-SLAM system will answer these questions to a certain extent.

Chapter 4

Implementation

4.1 System Setup

4.1.1 Hardware and Infrastructure

At first, it was decided to use Google Cloud service that provide the means to create, access and manipulate virtual. Particularly, *n1-standard-4* virtual machine was chosen. It has 4 virtual CPUs and 15 GB RAM which is surely enough for compilation of ORB-SLAM library. The operational system was chosen to be Ubuntu 18.04 LTS. The motivation behind this choice was that Google Cloud service gives a free trial with 300$ and researcher's laptop crashed whilst building ORB-SLAM2 library from source. Later this problem was solved; it turned out that ORB-SLAM's build script had inappropriate *Make* (building tool) call. After editing this script, the building process started to finish without crashing.

The laptop that was used is MSI GX60-1AC and it has following specifications: AMD A10-4600m **CPU**, 8GB DDR3 1600MHz **RAM**, AMD Radeon HD7970M **GPU**. Operating system that was used is Ubuntu 18.04 LTS. Further sections will emphasize what software packages and libraries were installed onto the laptop; all these packages are needed to be installed before starting the development of comparative application and ceres solver extensions.

MSI GX60-1AC	
Operating System	Ubuntu 18.04 LTS
CPU	AMD A10-4600m
GPU	Radeon HD7970M
RAM	8 GB DDR3

Table 4.1: MSI GX60 laptop specification

4.1.2 ORB-SLAM Installation Process

Before running any tests, or changing the code ORB-SLAM2 its dependencies should be installed. This subsection will describe the installation process on computer with linux system with aptitude packet manager. Most of these components were downloaded and installed from source code. Some packages were installed through aptitude package manager. List of the packages that should be installed:

1. First of all, the toolchain to compile C++ code should be installed; C++11 or C++0x compiler could be found in build-essentials package in aptitute package manager.

2. For visualization and user interface, ORB-SLAM uses Pangolin library that wraps over OpenGL [31].

3. The big part of the ORB-SLAM uses OpenCV functions that are mostly used for handling images, managing special data structures (for example for representing transnational and rotational coordinates). For the needs of the thesis project OpenCV 3.4.5 was used [32].

4. DBoW2 library [33] is used for indexing and converting images into a bag-of-word representation. It is already included in ORB-SLAM2 in Thirdparty folder.

5. G2o [10] is a graph optimization framework that includes vast amount of solvers and optimization methods of linear, nonlinear equations and graphs. It is also included in Thirdparty folder of ORB-SLAM source code.

6. Eigen3 is library for linear algebra that is used by g2o optimization framework [34]. It provides different data structures for handling matrices, vectors, numerical solvers and other algorithms.

When all dependencies are installed, ORB-SLAM itself needs to be installed. Root folder of the ORB-SLAM project contains **build.sh** script that compiles project according to CMake configuration file. After compilation of the project, there will be shared object libORB_SLAM2.so that will contain compiled source code of ORB_SLAM and its dependencies. The build process also compiles example programs that use ORB-SLAM2 library: mono_tum, mono_kitti, mono_euroc, stereo_euroc, stereo_kitti, rgbd_tum. Each of these is used for specific dataset which were described earlier in Chapters 2, 3.

4.1.3 QtCreator Installation

In order to develop Comparative application, proper GUI framework should be used. In methodology chapter 3 it was proposed to use Qt5 QWidgets. QtCreator is massive IDE that helps developers to create GUI applications easily as it include Designer feature that allows to create QWidget applications using

drag-and-drop. Then all the dependencies between forms, buttons and other elements should be written in C++. QtCreator's installer can be downloaded via their web-page [35].

4.2 Extending ORB-SLAM Optimization

4.2.1 ORB-SLAM Code Review

Having ORB-SLAM installed we need to find the place where we should change the code in order to test different optimization methods; also the code should be changed in specific places in order to use the ORB-SLAM dynamically compiled library with Comparison Application. Code project layout of ORB-SLAM library looks like this:

1. **Converter.h/.cpp**: Contains methods that allow to convert datastructures of one kind to another, e.g. g2o matrix to opencv matrix.
2. **Frame.h/.cpp**: Introduces camera frame object that has useful methods for manipulating frames.
3. **FrameDrawer.h/.cpp**: provides the means to draw frame with features.
4. **Initializer.h/.cpp**: initializer for tracking routine in monocular slam.
5. **KeyFrame.h/.cpp**: Data structure that encapsulates keyframes and has useful functions.
6. **KeyFrameDatabase.h/.cpp**: Data structure that allows to store keyframes. Also defines loop detection and re-localization algorithms.
7. **LocalMapping.h/.cpp**: local bundle adjustment in particular sliding window.
8. **LoopClosing.h/.cpp**: Contains implementation of loop closing algorithm.
9. **Map.h/.cpp**: Data structure for handling map of points and keyframes.
10. **MapDrawer.h/.cpp**: Means to draw the map.
11. **MapPoint.h/.cpp**: Data structure that handles a single map point.
12. **Optimizer.h/.cpp**: Contains several optimization solvers that we were looking for.
13. **ORBextractor.h/.cpp**: Feature extractor, given images.
14. **ORBmatcher.h/.cpp**: Point matcher between different frames.
15. **ORBVocabulary.h/.cpp**: Handles ORB Vocabulary file.

16. **PnPsolver.h/.cpp**: Contains Perspective-n-Point solver. PnP is the problem of estimating the pose of a calibrated camera given a set of 3D points in the world and their corresponding 2D projections in the image.

17. **Sim3Solver.h/.cpp**: Sim3 lie group defines 3D similarity transformations. Thus, this file defines the math behind Sim(3).

18. **System.h/.cpp**: Main file that is defines starting point of ORB-SLAM system.

19. **Tracking.h/.cpp**: Tracking routine that keeps track of points and refines them by solving optimization problems.

20. **Viewer.h/.cpp**: Uses Pangolin as the mean to display user interface.

The files that is of great interest are **Optimizer.cpp/.h**. Basically, it is the place where all optimization problems are formulated initially using g2o optimization framework. As it was already discussed, before there are five nonlinear optimization problems defined in ORB-SLAM2. They are represented as five corresponding functions in this file.

1. void static GlobalBundleAdjustment(): for Global Bundle Adjustment

2. void static LocalBundleAdjustment(): for Local Bundle Adjustment

3. int static PoseOptimization(): for optimization of poses of camera.

4. static int OptimizeSim3(): Optimization of transformation similarity between objects.

5. void static EssentialGraphOptimization(): Optimization of overall hypergraph.

Whilst working on thesis, only 4 of them were formulated using Ceres optimization framework, thus we will use in the final application only four choices: LocalBundleAdjustment, PoseOptimization, OptimizeSim3, and EssentialGraphOptimization. By default, every optimization solver function uses Levenberg-Marquardt method for solving nonlinear optimization problems.

Also, some changes need to be done in order to combine ORB-SLAM functionality with the functionality of Comparison application. Specifically, Comparison application produces JSON file with configurations of nonlinear optimization methods, then Comparison application launches the ORB-SLAM application process and feeds the JSON file to it. This configuration eventually propagates to the constructor of System object (that initializes all threads and starts SLAM), thus the appropriate changes to the System class should be made (parsing the JSON file and storing configurations in internal data structures in Optimizer. Optimizer then upon launching uses this parameter to perform the corresponding optimization).

Regarding the implementation of ceres-based extensions (optimization problems), the changes were made in MapPoint and KeyFrame classes by adding conversion mechanisms from one coordinate representation to Rodriguez (6 degree of freedom representation; 3 parameters for translation, 3 for rotation). Then of course the methods themselves were added to Optimizer class; corresponding error functions that are to be optimized were added. Converter class contains all the support functions that give the tools from converting data structures that hold information from one form to another;

4.2.2 Extension Implementation

In chapter 3 the models for implementing ceres-based optimization methods were proposed. The Ceres-based optimization problems are formulated almost the same as in g2o, but they are more configurable. As it was already mentioned in Literature Review chapter 2, first of all error function should be defined. Figure 4.1 shows the structures implemented in C++ that formulate error function (reprojection error in difference spaces and groups).

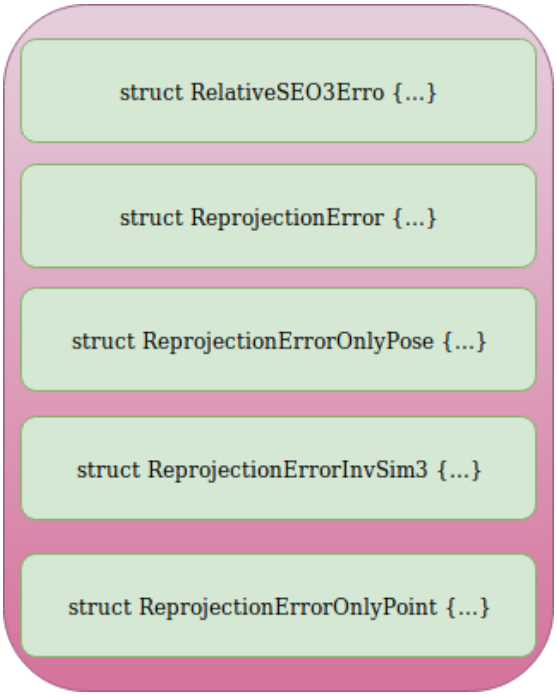


Figure 4.1: Structures that define error function which will be optimized in Ceres

In order to define the nonlinear optimization problem using Ceres solver appropriate loss functions and cost functions should defined. Loss function is usually called as robust kernel that handles outliers. In g2o loss function is either

Huber loss or it is absent at all. Cost functions are the error that were defined earlier. For each cost function differentiation module should be provided. In the case of Ceres solver it provides auto differentiation module that was used.

After implementing all needed structures, optimization problems should be reimplemented using Ceres solver instead of the g2o. Because of the fact that g2o optimization framework uses another way of solving nonlinear-optimization problems Ceres solver's implementation will be a little bit different. But overall the implementation of Ceres-based optimization problems is hugely inspired by g2o-based implementation of problems.

4.3 Comparison Framework Implementation

This section will describe implementation of the Comparison application that would comply the design discussed in chapter 3.

4.3.1 Application GUI

GUI is designed and implemented using Qt QWidgets and written in C++ programming language. The figure 4.2 shows the UML diagram for Comparison application. Classes that were implemented are:

- **MainWindow** - this class implements the main GUI element which allows user to adjust settings for current ORB-SLAM's configuration, e.g. set particular nonlinear optimization method, advanced configuration, etc. Main window has two buttons: first submits the task to the ORB-SLAM for execution, second switches window to the results window.

- **ResultsWindow** - this class implements the Results window which allows user to visualize trajectory, absolute pose error (APE) and look up previous results of experiments. It has 4 buttons: one for switching window back to the main window and 3 buttons for visualizing results.

- **AdvancedConfigCeres** - encapsulates the advanced settings for Ceres solver that could be set by user.

- **CeresTrustRegion** - It is a part of advanced configuration for Ceres. It stores fields for trust region algorithms.

- **CeresLineSearch** - it is also a part of advanced configuration for Ceres which stores fields and information about line search algorithms.

- **AdvancedConfigG2O** - it should encapsulate the advanced setting for G2O, but currently it stores nothing, because g2o optimization methods are only configured by method name in this thesis (Levenberg-Marquardt, Gauss-Newton or Dogleg).

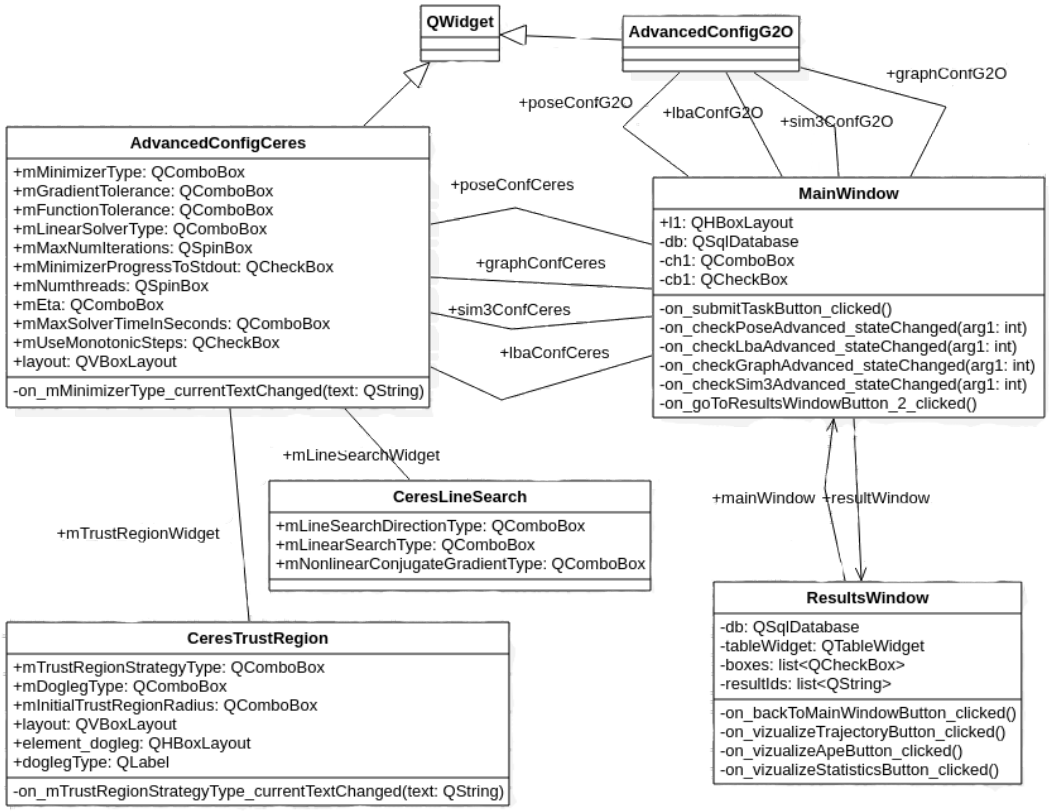


Figure 4.2: Final database design that is used for storing the information related to experiments

4.3.2 Database

The previous chapter discussed the high-level database architecture; implementation of such database should start from choosing the Database management system (DBMS). The final choice was the PostgreSQL that is very powerful open-source DBMS that allows developer to carefully design and implement needed database structure. Figure 4.3 defines the Database design that was used when implementing the Comparison application. *Libraries* table consist of all permutations of libraries for each method.

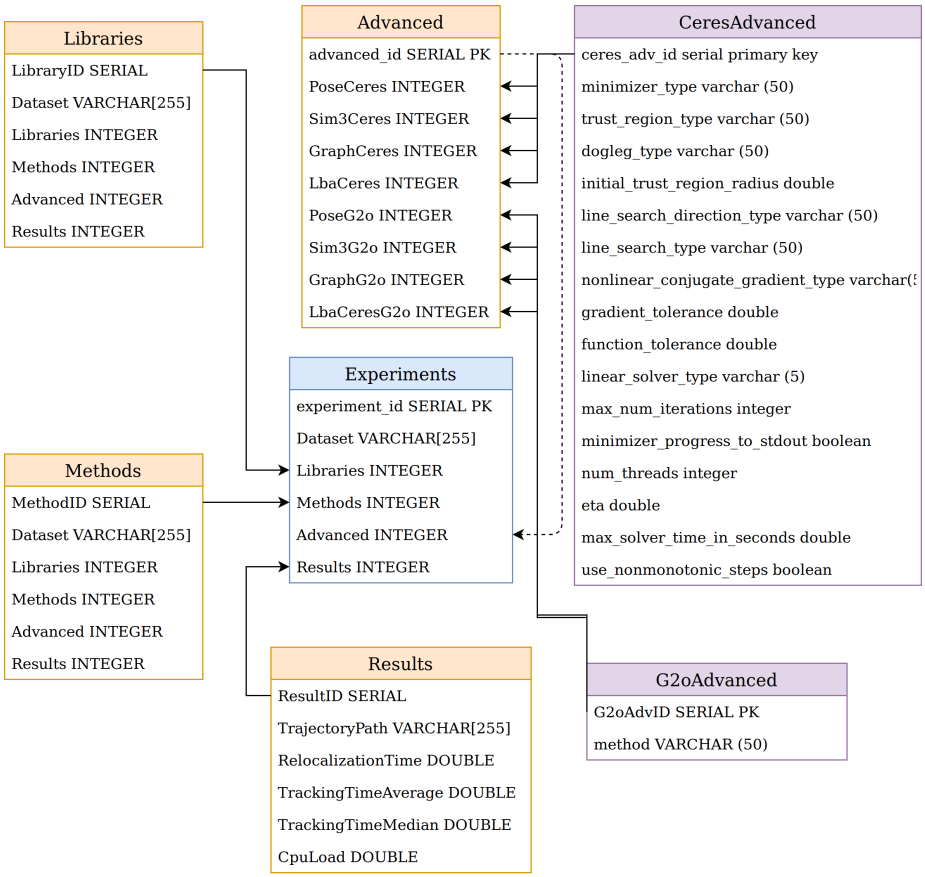


Figure 4.3: Final database design that is used for storing the information related to experiments

The code for creating database could be find in Appendix A.

Chapter 5

Evaluation and Discussion

Previous chapter described the implementation process of ORB-SLAM extensions and comparison framework. Now the evaluation of both pieces of work should be provided. This section will describe the work of the Comparison application and it will present the results that were gathered through that comparison framework. It will be seen how exactly configuration of non-linear optimization problems influence on the overall ORB-SLAM performance.

5.1 Comparison Application

This section will show the use case of how exactly to use the Comparison application step by step. Then the batch of experiments will be launched in order to get representative results that later be used for assessing different ORB-SLAM configurations; they will be compared using metric that were discussed previously in chapter 3, e.g. CPU consumption, time performance of various ORB-SLAM tasks in milliseconds, absolute pose error (APE) between generated trajectory and ground-truth (also called reference) trajectory, etc. This section is basically assess the GUI application itself: design issues and possible improvements will be discussed later.

5.1.1 Testing the application

This subsection will describe a typical use case that shows all the functionality of the Comparison application. The figure 5.1 shows the initial windows that give an opportunity to configure each of the 4 ORB-SLAM's optimization problem. After setting the optimization method of Essential graph optimization to g2o's Gauss-Newton, the experiment can be started by pressing the "Submit Task" button.

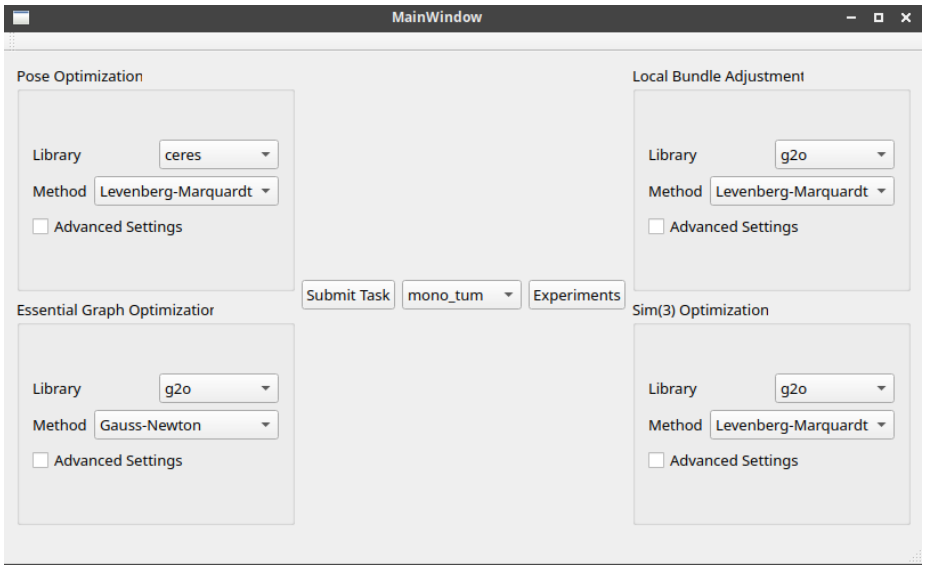


Figure 5.1: Initial main menu. The essential graph was chosen to be solved with Gauss-Newton g2o method.

After pressing the submit button, Comparison application collects json file with configuration and starts the ORB-SLAM application process. ORB-SLAM then initializes with the new configuration and launches visualization windows: first for viewing point cloud and trajectory, second for showing the tracked points on the current image. Figure 5.2 shows the visualization windows of ORB-SLAM.

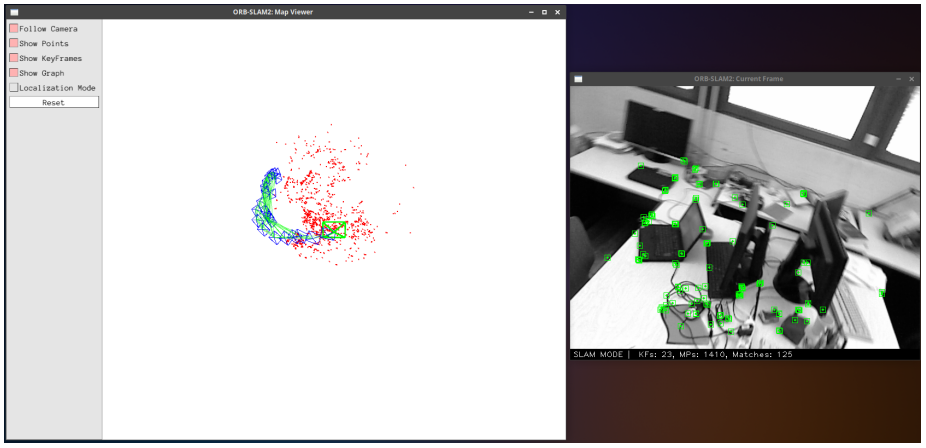


Figure 5.2: ORB-SLAM's visualization of tracking and localization

After ORB-SLAM finishes its work, the results are saved in json format and

Comparison application then takes them and using all the information about the experiment it had it inserts all of this information in the database. Now, the user can press the button "Experiments" on the main window which will get him to the Results window. Entries of the table correspond to the experiment.

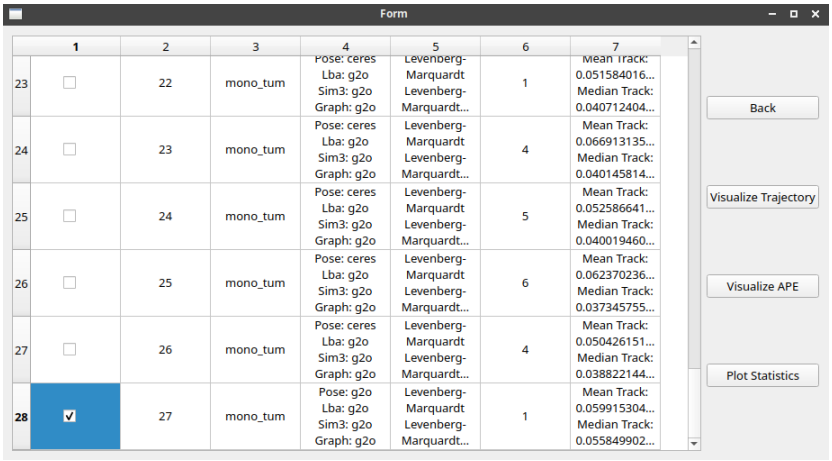
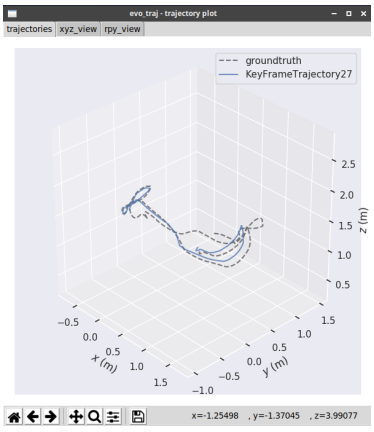
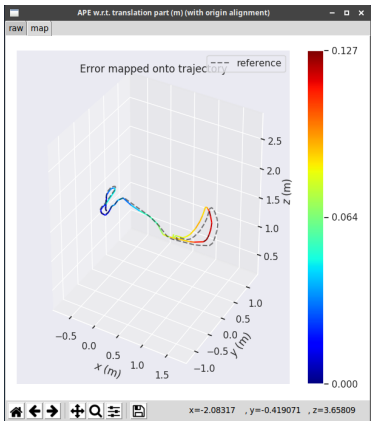


Figure 5.3: Second window that shows all the previous results and gives an opportunity to check the experiments and visualize their results

If user presses the button "Visualize Trajectory" on Results window, then the result of launching *evo_traj* with checked experiments will be shown. Specifically it will show checked trajectories compared to the ground truth trajectory.



(a) Plot that compares generated trajectory and ground truth trajectory



(b) Plot that shows absolute pose error between generated trajectory and reference trajectory

Figure 5.4: Plot the experiment

The figure 5.5 shows the main window that specifies the advanced setting for Ceres-based Pose optimization problem.

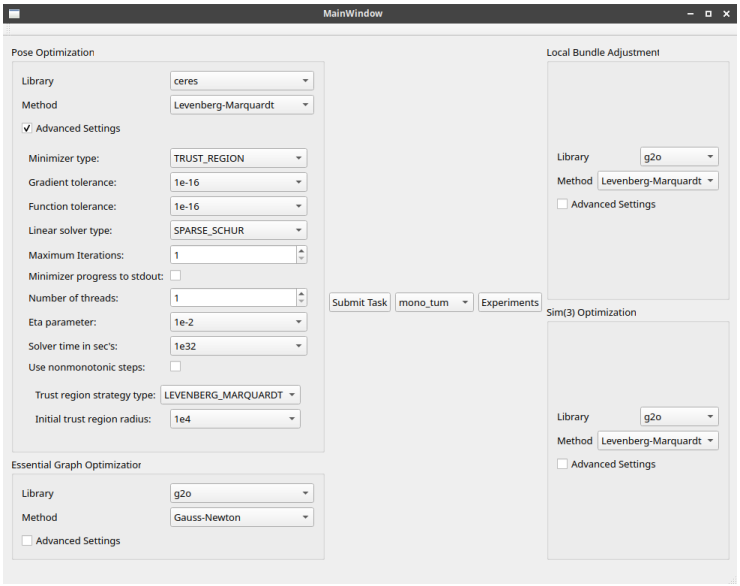


Figure 5.5: Initial main menu. Pose optimization is solved by ceres with advanced settings

5.2 Running Experiments

Now the experiments should be formulated. As it was described in methodology chapter 3 it is better to think of specific question that could be answered by Comparison application. Because plots for results take a lot of space, it was decided to take them out to Appendix chapter **??**.

5.2.1 Question 1

The question is "For mono_tum and for all optimization problems, which g2o combination of optimization methods is the best?". To try to answer this question 11 experiments were run. The figure 5.1 shows the results (median/mean tracking time).

5.3 Discussion of Results

As it can be seen the best result in terms of median and mean tracking time was shown by the first experiment which is the default settings for ORB-SLAM when all optimization methods are set to be Levenberg-Marquardt. But now the

id	Graph	Sim(3)	LBA	Pose	Mean (ms)	Median (ms)
1	LM	LM	LM	LM	58	55
2	GN	LM	LM	LM	59	58
3	DL	LM	LM	LM	59	58
4	LM	GN	LM	LM	59	55
5	LM	DL	LM	LM	62	61
6	LM	LM	GN	LM	63	62
7	LM	LM	DL	LM	74	71
8	LM	LM	LM	GN	60	59
9	LM	LM	LM	DL	76	79
10	GN	GN	GN	GN	Track Lost	Track Lost
11	DL	DL	DL	DL	Track Lost	Track Lost

Table 5.1: **LM** - Levenberg-Marquardt; **GN** - Gauss-Newton; **DL** - Dogleg; Green color stands for the best results in terms of median and mean tracking time

Absolute pose error should be visualized and compared between experiments. Results for Absolute pose error for question 1 could be seen in Figures B.1, B.2, and B.3. The experiment that has shown the best results is experiment 7 (the Figure B.3 show that it has the best results). Experiment 7 corresponds to the configuration of ORB-SLAM, when Local Bundle Adjustment is using g2o's Dogleg method and others optimization methods for corresponding problems are set to be Levenberg-Marquardt. There was a problem with launching Ceres-based optimization problems - they after a while lost the track and didn't produce the desired trajectory. This problem can be solved in the future.

Chapter 6

Conclusion

6.1 Conclusion

This thesis provided the research of how ORB-SLAM's configuration of nonlinear optimization methods impacts its the performance. During the course of the thesis development next significant results were obtained:

- **Extending ORB-SLAM nonlinear methods**: As the part of the research of this thesis, it was decided to extend ORB-SLAM with Ceres-based optimization problems. This could extend the scope of the research, by allowing to compare two optimization frameworks and deciding which framework impacts on the performance of the SLAM system stronger.

- **Design and implementation of Comparison application**: One of the biggest parts of this thesis is Comparison application. It allows testing different configurations of ORB-SLAM, e.g. run ORB-SLAM on particular dataset with specific configuration of nonlinear optimization methods, showing the evaluation results (mean/median tracking time, absolute pose error), visualizing plots. Implementation of Comparison application involved designing and implementing Qt based GUI application, designing and deploying the database for storing the results, extending ORB-SLAM library to work with Comparison application.

- **Running experiments**: Several experiments were run in order to test Comparison application and to find out which configuration of nonlinear optimization methods was optimal for given question. For example, for question 1 results has shown that the configuration where local bundle adjustment is solved using dogleg optimization method performs better than others in terms of Absolute pose error. The default configuration for g2o-based optimization problems (all problems are solved using Levenberg-Marquardt) is found the best in terms of mean/median tracking time, but it generates inaccurate trajectory with high absolute pose error.

6.2 Future works

This topic of comparing nonlinear optimization methods can be further explored; more optimization techniques could be implemented using different optimization frameworks.

Comparison application can be further extended with new functionalities like:

- Add several experiments into the batch before starting the experiments; this would decrease the time user spends on configuring and running separate experiments.
- Add permutation flag to each property (like optimization method), which would create specific number of configurations with each permutation. This would made testing multiple configurations more automated and comfortable for the user to work with.

Bibliography

[1] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[2] R. Mur-Artal, J. D. Tardos, J. M. M. Montiel, and D. Galvez-Lopez, "Real-time slam for monocular, stereo and rgb-d cameras, with loop detection and relocalization capabilities," https://github.com/raulmur/ORB_SLAM2.

[3] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[4] L. Khachiyan, "A polynomial algorithm in linear programming," *Soviet Mathematics Doklady*, vol. 20, pp. 191–194, 1979.

[5] N. Karmarkar, "A new polynomial-time algorithm for linear programming," *Combinatorics*, vol. 4, pp. 373–395, 1984.

[6] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 2013.

[7] K. Madsen, H. Nielsen, and O. Tingleff, *Methods for nonlinear least squares problems*, 2004.

[8] S. J. Wright and J. N. Holt, "An inexact levenberg marquardt method for large sparse nonlinear least squares," *Journal of the Australian Mathematical Society Series B*, vol. 4, pp. 387–403, 1985.

[9] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g 2 o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3607–3613.

[10] R. Kuemmerle and et al., "g2o: A general framework for graph optimization," https://github.com/RainerKuemmerle/g2o.

[11] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics Automation Magazine*, vol. 13, pp. 99–110, 2006.

[12] A. Das, "Nonlinear state estimation overview," http://wavelab.uwaterloo.ca/slam/2017-SLAM/Lecture1-Overview/Reading%20Course%20Overview%20-%20Revised.pdf.

[13] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[14] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1052–1067, 2007.

[15] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," *Intl. Symposium on Mixed and Augmented Reality*, 2007.

[16] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun, "Dense monocular depth estimation in complex dynamic scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 4058–4066.

[17] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*. IEEE, 2011, pp. 2320–2327.

[18] M. Pizzoli, C. Forster, and D. Scaramuzza, "Remode: Probabilistic, monocular dense reconstruction in real time," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 2609–2616.

[19] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European conference on computer vision*. Springer, 2014, pp. 834–849.

[20] S. Leutenegger, P. Furgale, V. Rabaud, M. Chli, K. Konolige, and R. Siegwart, "Keyframe-based visual-inertial slam using nonlinear optimization," *Proceedings of Robotis Science and Systems (RSS) 2013*, 2013.

[21] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustmenta modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

[22] H. Liu, M. Chen, Y. Bao, and Z. Wang, "Ice-ba: Incremental, consistent and efficient bundle adjustment for visual-inertial slam," https://github.com/baidu/ICE-BA.

[23] A. I. Mourikis and S. I. Roumeliotis, "A multi- state constraint kalman filter for vision-aided inertial navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, 2007.

[24] E. S. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *International Journal of Robotics Research*, vol. 30, pp. 407–430, 2011.

[25] "Technical university of munich (tum) datasets for slam," https://vision.in.tum.de/data/datasets.

[26] J. Sturm, W. Burgard, and D. Cremers, "Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark," in *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[27] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[28] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[29] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[30] M. Grupp, "Evo: Python package for the evaluation of odometry and slam," https://github.com/MichaelGrupp/evo.

[31] S. Lovegrove, "Pangolin," https://github.com/stevenlovegrove/Pangolin.

[32] "Opencv," https://opencv.org/releases.html.

[33] "Dbow: Bag of words," https://github.com/dorian3d/DBoW2.

[34] "Eigen: Linear algebra library," http://eigen.tuxfamily.org.

[35] "Qtcreator ide," https://www.qt.io/download.

Appendix A

Database

A.1 Queries for creating the database

```
CREATE TABLE libraries(
library_id SERIAL PRIMARY KEY,
pose_optim_lib VARCHAR (30) NOT NULL,
lba_optim_lib VARCHAR (30) NOT NULL,
sim3_optim_lib VARCHAR (30) NOT NULL, graph_optim_lib VARCHAR (30)
    NOT NULL);


 CREATE TABLE methods(
method_id SERIAL PRIMARY KEY,
pose_optim_method VARCHAR (60) NOT NULL,
lba_optim_method VARCHAR (60) NOT NULL,
sim3_optim_method VARCHAR (60) NOT NULL, graph_optim_method VARCHAR
     (60) NOT NULL);

CREATE TABLE results(
result_id SERIAL PRIMARY KEY,
trajectory_path VARCHAR (255) NOT NULL,
tracking_time_average DOUBLE PRECISION NOT NULL,
tracking_time_median DOUBLE PRECISION NOT NULL);

CREATE TABLE advanced_ceres(
ceres_adv_id SERIAL PRIMARY KEY,
minimizer_type VARCHAR (50) NOT NULL,
trust_region_type VARCHAR (50),
dogleg_type VARCHAR (50),
initial_trust_region_radius DOUBLE PRECISION,
line_search_direction_type VARCHAR (50),
line_search_type VARCHAR(50),
nonlinear_conjugate_gradient_type VARCHAR(50),
gradient_tolerance DOUBLE PRECISION NOT NULL,
function_tolerance DOUBLE PRECISION NOT NULL,
linear_solver_type VARCHAR (50) NOT NULL,
max_num_iterations INTEGER NOT NULL,
minimizer_progress_to_stdout BOOLEAN NOT NULL,
num_threads INTEGER NOT NULL,
```

```
35 eta DOUBLE PRECISION NOT NULL,
36 max_solver_time_in_seconds DOUBLE PRECISION NOT NULL,
37 use_nonmonotonic_steps BOOLEAN NOT NULL);
38
39 INSERT INTO advanced_ceres (minimizer_type, trust_region_type,
       initial_trust_region_radius, gradient_tolerance,
       function_tolerance, linear_solver_type, max_num_iterations,
40                 minimizer_progress_to_stdout, num_threads, eta,
       max_solver_time_in_seconds, use_nonmonotonic_steps) VALUES ('
       TRUST_REGION', 'LEVENBERG_MARQUARDT', 1e4, 1e-16, 1e-16, '
       SPARSE_SCHUR', 5, TRUE, 1, 1e-2, 1e32, TRUE) returning
       ceres_adv_id;
41
42 CREATE TABLE advanced_g2o(
43 g2o_adv_id SERIAL PRIMARY KEY,
44 method VARCHAR (50) NOT NULL);
45
46 INSERT INTO advanced_g2o (method) VALUES ('levenberg-marquardt');
47 INSERT INTO advanced_g2o (method) VALUES ('gauss-newton');
48 INSERT INTO advanced_g2o (method) VALUES ('dogleg');
49
50 CREATE TABLE advanced(
51 advanced_id SERIAL PRIMARY KEY,
52 pose_ceres INTEGER REFERENCES advanced_ceres,
53 pose_g2o INTEGER REFERENCES advanced_g2o,
54 lba_ceres INTEGER REFERENCES advanced_ceres,
55 lba_g2o INTEGER REFERENCES advanced_g2o,
56 sim3_ceres INTEGER REFERENCES advanced_ceres,
57 sim3_g2o INTEGER REFERENCES advanced_g2o,
58 graph_ceres INTEGER REFERENCES advanced_ceres,
59 graph_g2o INTEGER REFERENCES advanced_g2o);
60
61 CREATE TABLE experiments(
62 experiment_id SERIAL PRIMARY KEY,
63 dataset_name VARCHAR (60) NOT NULL,
64 libraries INTEGER REFERENCES libraries,
65 methods INTEGER REFERENCES methods,
66 advanced INTEGER REFERENCES advanced,
67 results INTEGER REFERENCES results ON DELETE CASCADE);
68
69 INSERT INTO libraries (pose_optim_lib, lba_optim_lib,
       sim3_optim_lib, graph_optim_lib) VALUES ('g2o', 'g2o', 'g2o', '
       g2o'),
70   ('g2o', 'g2o', 'g2o', 'ceres'),
71   ('g2o', 'g2o', 'ceres', 'g2o'),
72   ('g2o', 'g2o', 'ceres', 'ceres'),
73   ('g2o', 'ceres', 'g2o', 'g2o'),
74   ('g2o', 'ceres', 'g2o', 'ceres'),
75   ('g2o', 'ceres', 'ceres', 'g2o'),
76   ('g2o', 'ceres', 'ceres', 'ceres'),
77   ('ceres', 'g2o', 'g2o', 'g2o'),
78   ('ceres', 'g2o', 'g2o', 'ceres'),
79   ('ceres', 'g2o', 'ceres', 'g2o'),
80   ('ceres', 'g2o', 'ceres', 'ceres'),
81   ('ceres', 'ceres', 'g2o', 'g2o'),
82   ('ceres', 'ceres', 'g2o', 'ceres'),
83   ('ceres', 'ceres', 'ceres', 'g2o'),
```

```
84    ('ceres', 'ceres', 'ceres', 'ceres');
85
86  INSERT INTO methods (pose_optim_method, lba_optim_method,
        sim3_optim_method, graph_optim_method) VALUES ('Levenberg-
        Marquardt', 'Levenberg-Marquardt', 'Levenberg-Marquardt', '
        Levenberg-Marquardt'),
87    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Levenberg-
        Marquardt', 'Gauss-Newton'),
88    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Levenberg-
        Marquardt', 'Dogleg'),
89    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Gauss-Newton', '
        Levenberg-Marquardt'),
90    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Gauss-Newton', '
        Gauss-Newton'),
91    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Gauss-Newton', '
        Dogleg'),
92    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Dogleg', '
        Levenberg-Marquardt'),
93    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Dogleg', 'Gauss-
        Newton'),
94    ('Levenberg-Marquardt', 'Levenberg-Marquardt', 'Dogleg', 'Dogleg'
        ),
95    ('Levenberg-Marquardt', 'Gauss-Newton', 'Levenberg-Marquardt', '
        Levenberg-Marquardt'),
96    ('Levenberg-Marquardt', 'Gauss-Newton', 'Levenberg-Marquardt', '
        Gauss-Newton'),
97    ('Levenberg-Marquardt', 'Gauss-Newton', 'Levenberg-Marquardt', '
        Dogleg'),
98    ('Levenberg-Marquardt', 'Gauss-Newton', 'Gauss-Newton', '
        Levenberg-Marquardt'),
99    ('Levenberg-Marquardt', 'Gauss-Newton', 'Gauss-Newton', 'Gauss-
        Newton'),
100   ('Levenberg-Marquardt', 'Gauss-Newton', 'Gauss-Newton', 'Dogleg')
        ,
101   ('Levenberg-Marquardt', 'Gauss-Newton', 'Dogleg', 'Levenberg-
        Marquardt'),
102   ('Levenberg-Marquardt', 'Gauss-Newton', 'Dogleg', 'Gauss-Newton')
        ,
103   ('Levenberg-Marquardt', 'Gauss-Newton', 'Dogleg', 'Dogleg'),
104   ('Levenberg-Marquardt', 'Dogleg', 'Levenberg-Marquardt', '
        Levenberg-Marquardt'),
105   ('Levenberg-Marquardt', 'Dogleg', 'Levenberg-Marquardt', 'Gauss-
        Newton'),
106   ('Levenberg-Marquardt', 'Dogleg', 'Levenberg-Marquardt', 'Dogleg'
        ),
107   ('Levenberg-Marquardt', 'Dogleg', 'Gauss-Newton', 'Levenberg-
        Marquardt'),
108   ('Levenberg-Marquardt', 'Dogleg', 'Gauss-Newton', 'Gauss-Newton')
        ,
109   ('Levenberg-Marquardt', 'Dogleg', 'Gauss-Newton', 'Dogleg'),
110   ('Levenberg-Marquardt', 'Dogleg', 'Dogleg', 'Levenberg-Marquardt'
        ),
111   ('Levenberg-Marquardt', 'Dogleg', 'Dogleg', 'Gauss-Newton'),
112   ('Levenberg-Marquardt', 'Dogleg', 'Dogleg', 'Dogleg'),
113   ('Gauss-Newton', 'Levenberg-Marquardt', 'Levenberg-Marquardt', '
        Levenberg-Marquardt'),
114   ('Gauss-Newton', 'Levenberg-Marquardt', 'Levenberg-Marquardt', '
```

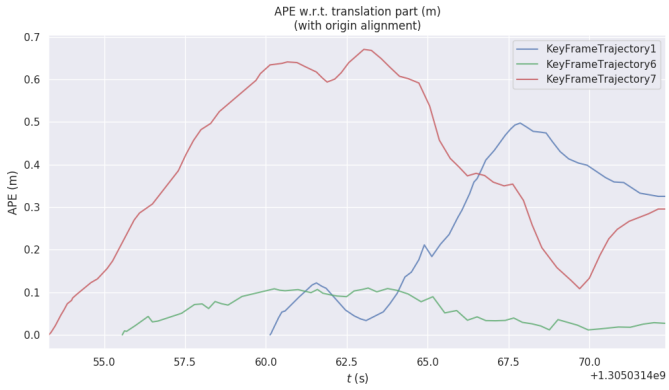
```
    Gauss-Newton'),
115 ('Gauss-Newton', 'Levenberg-Marquardt', 'Levenberg-Marquardt', '
    Dogleg'),
116 ('Gauss-Newton', 'Levenberg-Marquardt', 'Gauss-Newton', '
    Levenberg-Marquardt'),
117 ('Gauss-Newton', 'Levenberg-Marquardt', 'Gauss-Newton', 'Gauss-
    Newton'),
118 ('Gauss-Newton', 'Levenberg-Marquardt', 'Gauss-Newton', 'Dogleg')
    ,
119 ('Gauss-Newton', 'Levenberg-Marquardt', 'Dogleg', 'Levenberg-
    Marquardt'),
120 ('Gauss-Newton', 'Levenberg-Marquardt', 'Dogleg', 'Gauss-Newton')
    ,
121 ('Gauss-Newton', 'Levenberg-Marquardt', 'Dogleg', 'Dogleg'),
122 ('Gauss-Newton', 'Gauss-Newton', 'Levenberg-Marquardt', '
    Levenberg-Marquardt'),
123 ('Gauss-Newton', 'Gauss-Newton', 'Levenberg-Marquardt', 'Gauss-
    Newton'),
124 ('Gauss-Newton', 'Gauss-Newton', 'Levenberg-Marquardt', 'Dogleg')
    ,
125 ('Gauss-Newton', 'Gauss-Newton', 'Gauss-Newton', 'Levenberg-
    Marquardt'),
126 ('Gauss-Newton', 'Gauss-Newton', 'Gauss-Newton', 'Gauss-Newton'),
127 ('Gauss-Newton', 'Gauss-Newton', 'Gauss-Newton', 'Dogleg'),
128 ('Gauss-Newton', 'Gauss-Newton', 'Dogleg', 'Levenberg-Marquardt')
    ,
129 ('Gauss-Newton', 'Gauss-Newton', 'Dogleg', 'Gauss-Newton'),
130 ('Gauss-Newton', 'Gauss-Newton', 'Dogleg', 'Dogleg'),
131 ('Gauss-Newton', 'Dogleg', 'Levenberg-Marquardt', 'Levenberg-
    Marquardt'),
132 ('Gauss-Newton', 'Dogleg', 'Levenberg-Marquardt', 'Gauss-Newton')
    ,
133 ('Gauss-Newton', 'Dogleg', 'Levenberg-Marquardt', 'Dogleg'),
134 ('Gauss-Newton', 'Dogleg', 'Gauss-Newton', 'Levenberg-Marquardt')
    ,
135 ('Gauss-Newton', 'Dogleg', 'Gauss-Newton', 'Gauss-Newton'),
136 ('Gauss-Newton', 'Dogleg', 'Gauss-Newton', 'Dogleg'),
137 ('Gauss-Newton', 'Dogleg', 'Dogleg', 'Levenberg-Marquardt'),
138 ('Gauss-Newton', 'Dogleg', 'Dogleg', 'Gauss-Newton'),
139 ('Gauss-Newton', 'Dogleg', 'Dogleg', 'Dogleg'),
140 ('Dogleg', 'Levenberg-Marquardt', 'Levenberg-Marquardt', '
    Levenberg-Marquardt'),
141 ('Dogleg', 'Levenberg-Marquardt', 'Levenberg-Marquardt', 'Gauss-
    Newton'),
142 ('Dogleg', 'Levenberg-Marquardt', 'Levenberg-Marquardt', 'Dogleg'
    ),
143 ('Dogleg', 'Levenberg-Marquardt', 'Gauss-Newton', 'Levenberg-
    Marquardt'),
144 ('Dogleg', 'Levenberg-Marquardt', 'Gauss-Newton', 'Gauss-Newton')
    ,
145 ('Dogleg', 'Levenberg-Marquardt', 'Gauss-Newton', 'Dogleg'),
146 ('Dogleg', 'Levenberg-Marquardt', 'Dogleg', 'Levenberg-Marquardt'
    ),
147 ('Dogleg', 'Levenberg-Marquardt', 'Dogleg', 'Gauss-Newton'),
148 ('Dogleg', 'Levenberg-Marquardt', 'Dogleg', 'Dogleg'),
149 ('Dogleg', 'Gauss-Newton', 'Levenberg-Marquardt', 'Levenberg-
    Marquardt'),
```

```
150     ('Dogleg', 'Gauss-Newton', 'Levenberg-Marquardt', 'Gauss-Newton')
          ,
151     ('Dogleg', 'Gauss-Newton', 'Levenberg-Marquardt', 'Dogleg'),
152     ('Dogleg', 'Gauss-Newton', 'Gauss-Newton', 'Levenberg-Marquardt')
          ,
153     ('Dogleg', 'Gauss-Newton', 'Gauss-Newton', 'Gauss-Newton'),
154     ('Dogleg', 'Gauss-Newton', 'Gauss-Newton', 'Dogleg'),
155     ('Dogleg', 'Gauss-Newton', 'Dogleg', 'Levenberg-Marquardt'),
156     ('Dogleg', 'Gauss-Newton', 'Dogleg', 'Gauss-Newton'),
157     ('Dogleg', 'Gauss-Newton', 'Dogleg', 'Dogleg'),
158     ('Dogleg', 'Dogleg', 'Levenberg-Marquardt', 'Levenberg-Marquardt'
          ),
159     ('Dogleg', 'Dogleg', 'Levenberg-Marquardt', 'Gauss-Newton'),
160     ('Dogleg', 'Dogleg', 'Levenberg-Marquardt', 'Dogleg'),
161     ('Dogleg', 'Dogleg', 'Gauss-Newton', 'Levenberg-Marquardt'),
162     ('Dogleg', 'Dogleg', 'Gauss-Newton', 'Gauss-Newton'),
163     ('Dogleg', 'Dogleg', 'Gauss-Newton', 'Dogleg'),
164     ('Dogleg', 'Dogleg', 'Dogleg', 'Levenberg-Marquardt'),
165     ('Dogleg', 'Dogleg', 'Dogleg', 'Gauss-Newton'),
166     ('Dogleg', 'Dogleg', 'Dogleg', 'Dogleg');
```

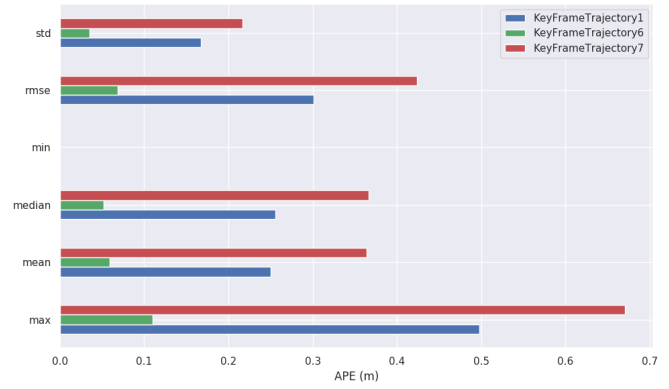
Appendix B

Plots and visualizations

B.1 Question 1: Results

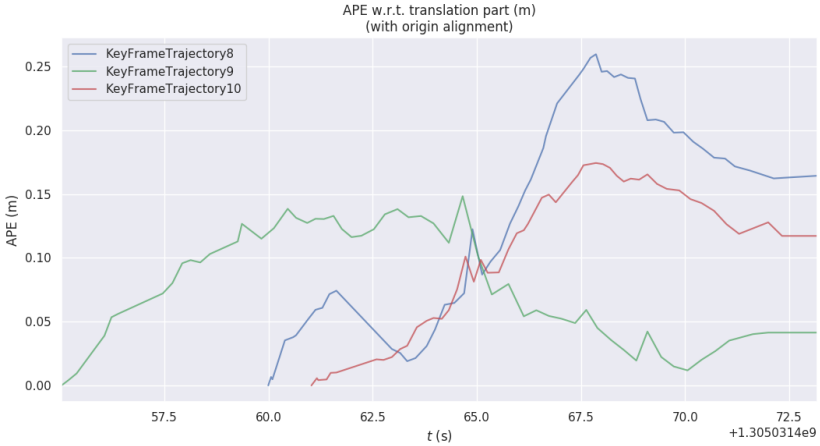


(a) APE: **KeyFrameTrajectory1** - results for first experiment (all LM); **KeyFrameTrajectory6** - result for second experiment (graph - GN); **KeyFrameTrajectory7** - result for third experiment (graph - DL)

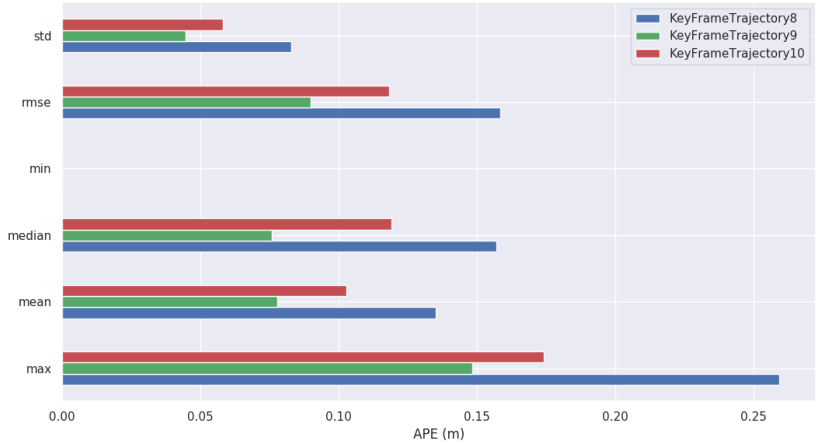


(b) Other metrics; root mean square error is the best for second experiment

Figure B.1: Results for experiments 1, 2, 3 for question 1

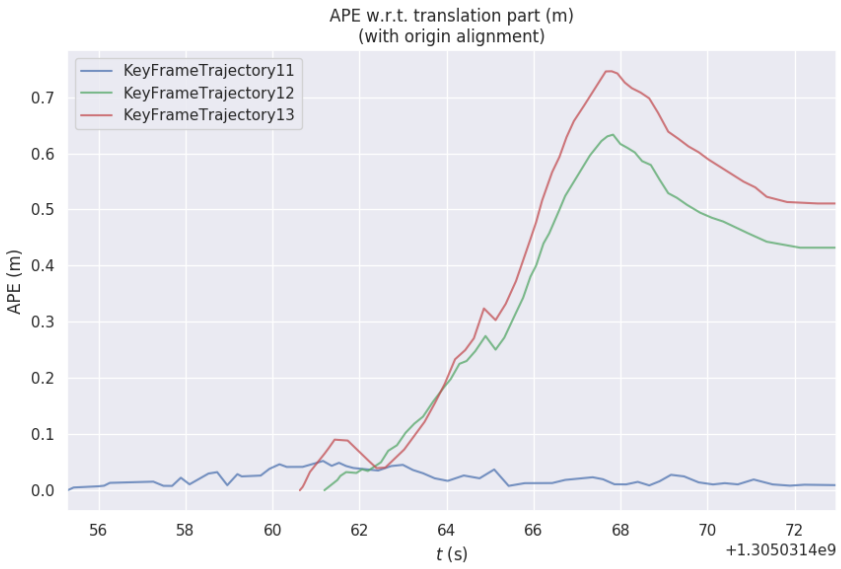


(a) APE: **KeyFrameTrajectory8** - results for fourth experiment (sim(3) - GN); **KeyFrameTrajectory9** - result for fifth experiment (sim(3) - DL); **KeyFrameTrajectory10** - result for sixth experiment (LBA - GN)

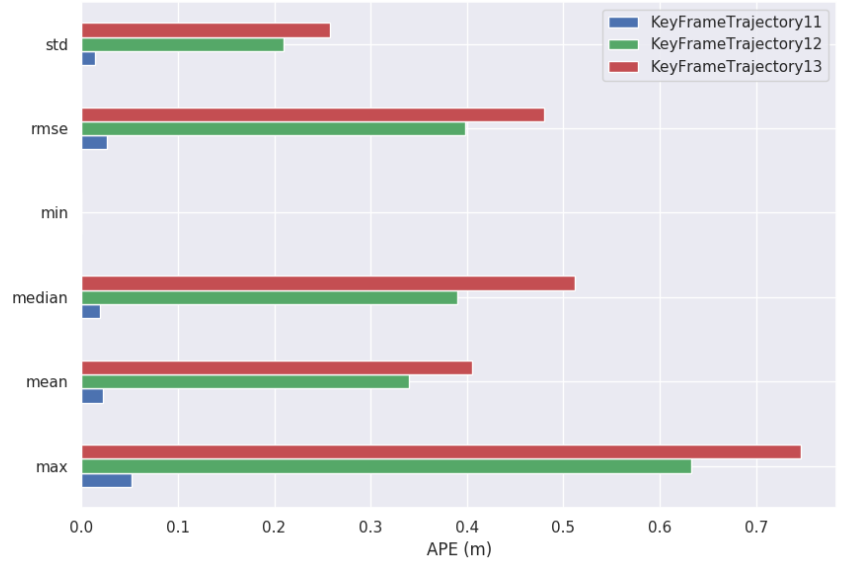


(b) Other metrics

Figure B.2: Results for experiments 4, 5, 6 for question 1



(a) APE: **KeyFrameTrajectory11** - results for seventh experiment (LBA - DL); **KeyFrameTrajectory12** - result for eighth experiment (Pose - GN); **KeyFrameTrajectory13** - result for ninth experiment (Pose - DL)



(b) Other metrics

Figure B.3: Results for experiments 7, 8 and 9 for question 1