University of Birmingham

School of Computer Science

MSc Robotics

Project Report

Graph-Based Visual SLAM for a 6DoF Robot

Author:
Saif Sidhik (1734904)

Supervisor:
Dr. Jeremy L Wyatt

September 11, 2017

Abstract

This report presents the development, implementation, testing and results thereof of a graph-based Simultaneous Localisation and Mapping (SLAM) system for a 6 degrees-of-freedom robot using just the on-board monocular camera. The project aims to develop a framework for an aerial robot to create a map of the environment as it traverses through it while simultaneously localising itself within the generated map.

The developed algorithm uses the sequence of images obtained from the camera to solve the SLAM problem offline. The graph-based approach adopted for this project makes use of the graphical representation of SLAM, wherein robot poses and positions of landmarks in the map become nodes of the graph and the measurements from the sensor form the constraints between them. These constraints or edges are obtained from observations of the environment or from movement actions taken by the robot. The graph is then optimised so as to obtain the configuration of nodes (robot poses and landmark positions) that would best fit the measurements made.

As the algorithm adds constraints to the graph when loop closures (previously mapped regions in the map) are re-observed, detecting false loop closures cause irrecoverable errors in the map and trajectory estimation. A novel way of filtering out false loop closure detections is presented in the report, using 'Optical Flow Check' as the final verification step. These modifications (made to the standard FAB-MAP loop detection framework) are shown in this report to be a fail-safe way of filtering out false detections. Optimising the graph using true loop closures has been shown to produce remarkable improvements in the final map and trajectory estimate.

Keywords: SLAM, full SLAM, graph-based SLAM, visual SLAM, monocular SLAM, loop closure detection, constraint optimisation.

Acknowledgements

Firstly, I would like to thank Dr. Jeremy L Wyatt for his supervision on this project, and for all his timely advices and suggestions that helped me manage and organise the project. I am also extremely indebted to Ermano Arruda, without whose help and guidance this project would not have been possible. I would like to thank him for the numerous hours he had spent to help me with the theory involved and tackle the countless practical difficulties that had come up throughout the project development, and for the many references and advices that he had provided me with, all of which were crucial in the development of this project. I would also like to thank Diar Karim for providing me with the hardware required, for getting me 24-hour-access to the Posture & Balance Lab where all the datasets for the project were created, and for all his support and motivation which were monumental in the completion of this project. Finally, I would like to thank Milan and Sam for the music and laughter that gave colour to the grey days.

Declaration

The material contained within this thesis has not previously been submitted for a degree at the University of Birmingham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Signed .....

Contents

List of Figures

Chapter 1

Introduction

Being able to create a map of the environment and to simultaneously know its own position within the map is an essential skill for mobile robots navigating in unknown environments, especially in the absence of external referencing systems such as GPS. This so-called Simultaneous Localization and Mapping (SLAM) problem has been one of the most studied topics in mobile robotics for the last few decades. Many solutions have been developed and described in literature to solve the SLAM problem.

SLAM approaches have been categorized into filtering approaches and smoothing approaches. Filtering approaches are concerned with solving the online SLAM problem in which only the current robot state and the map are estimated by incorporating sensor measurements as soon as they become available. On the other hand, smoothing (or offline) approaches address the full SLAM problem in which the posterior is estimated over the entire path along with the map, and is generally solved using least square error minimisation techniques [1]. As batch processing (the full SLAM approach) provides the most accurate and robust solution to any estimation problem in applications [2], a graph-based smoothing approach was developed and tested for this project. It attempts to solve the SLAM problem using a six degrees-of-freedom (6-DoF) aerial robot with only one input sensor – a monocular camera.

SLAM aims to obtain global consistency in the trajectory estimate and map. This is achieved by realizing that a previously mapped area has been re-visited (loop closure), and then updating the beliefs using this information to reduce the drifts in the estimates. Loop closure detection is the final refinement step and is vital for obtaining a globally consistent SLAM solution especially when localizing and mapping over long periods of time [3]. At the same time, detecting false loop closures could lead to very wrong map and trajectory estimations. Performance of graph-based SLAM algorithms is highly affected by the accuracy of the loop closures detected. One false loop closure could lead to errors in mapping that are potentially irrecoverable [4]. Therefore, avoiding such false detections is absolutely vital. A novel approach to verify the validity of loop closures was developed for this project. Its performance was compared with the standard FAB-MAP (Fast Appearance Based Mapping) [5, 6] loop closure detection framework. The modifications made were able to bring down the false loop closure detections made by the FAB-MAP algorithm to zero, at the cost of a small decrease in the number of true positives. The performance of the algorithm after optimising the graph with correct and wrong loop closures have been compared and analysed.

A video demonstrating the performance of the developed SLAM system is available (Appendix B). It demonstrates the main features of the algorithm, and shows the code running on various original and synthetic datasets. It also shows the different situations where the algorithm was found to fail.

The remainder of the report is organised as follows: Chapter 2 describes the gen-

eral SLAM problem in its probabilistic formulation, and introduces some relevant topics associated with SLAM. The theory and the mathematics of the constraint optimisation procedure involved in the graph-based SLAM approach is discussed in Chapter 3. Chapter 4 explains how the SLAM problem is generally approached using camera and vision. The hardware and methodology involved in the development and implementation of the graph-based visual SLAM system is explained in Chapter 5. The algorithm was tested on three datasets that were specifically created for the project. The testing of the system on these datasets and their results are presented in Chapter 6. Chapter 7 evaluates the performance of the algorithm and discusses possible improvements that could be made to tackle its shortcomings. The results are summarized and the report concluded in Chapter 8.

Chapter 2

The SLAM Problem

A representation of the environment, i.e., a map, is essential for many robot tasks such as transportation, manipulation, navigation, and almost all tasks that involve interaction with the environment. The process of computing such a representation from the data gathered by the sensors on the robot is referred to as robot spacial mapping. This problem has been studied since the creation of the first autonomous mobile robot, SHAKEY [7], almost half a century ago.

By solving the **mapping** problem, the robot discovers its environment while moving in it. This necessitates the regions perceived from a given location to be related to previously observed ones, and integrated together consistently. The result of this integration is a map representing the layout of the environment traversed by the robot. The term mapping in the context of this report will mean the process of gathering complex information from a sensor and subsequently representing it in an abstract and concise manner.

It is necessary to know (or estimate) the correct transformation between the newly observed region and the already built map, so that the observations can be integrated reliably to form a map. This transformation can be obtained by relating the current position of the robot to the previous ones. Therefore, proper **localisation** of the robot is necessary to properly relate the newly perceived regions with the already known regions.

In the absence of some external method of localisation such as GPS or motion capture systems which would give a strong constraint on the mapping, there are two ways to localize the robot [8]. One is incremental integration of robot motion, such as odometry or inertial units, which is known to eventually drift due to error accumulation and integration. The other method is to detect known features in the environment and use them as landmarks. However, this means having the map of the environment (atleast the positions of a few of the landmarks). Therefore, incremental environment mapping and robot localisation become two intimately related processes. The problem to be solved now becomes **Simultaneous Localisation and Mapping (SLAM)**. The general steps in a SLAM process follows the steps depicted in Figure 2.1.

Problem Definition

The SLAM problem can be defined as follows: A mobile robot starts at a location with known coordinates, and moves in an unknown environment. Its motion is uncertain, making it more and more difficult to determine its global coordinates over time. The robot can sense the environment as it moves in the environment. The SLAM problem is the problem of building a map of the environment while simultaneously determining the robot's position relative to the map. The map is assumed to be populated by observable features called landmarks that the robot can recognize and distinguish.

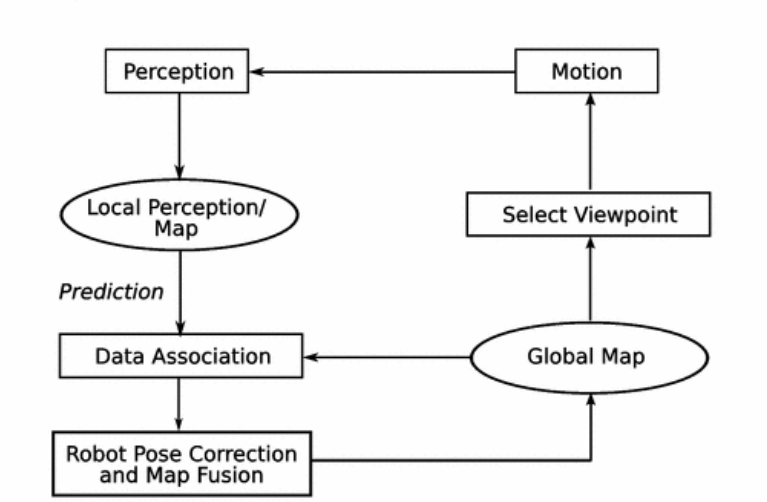


Figure 2.1: General Slam Process
Source: [8]

Mathematical Basis

It became very clear since the mid eighties that uncertainties in data perception and integration was central in the mapping process [8]. Sensors are never perfect; their data are noisy, inaccurate and incomplete. Therefore SLAM is best described in a probabilistic format.

Following the symbol conventions used by S. Thrun in his publications [9, 10], let the robot location at time t be denoted by x_t. In case of a mobile robot on a flat ground, x_t is usually a three-dimensional vector comprising of two values for denoting its 2D coordinates in the plane and one value for its orientation. For an aerial robot with no restrictions in its motion, x_t will be at least six-dimensional - three values for its 3D coordinate position and a minimum of three values for denoting its orientation. The sequence of locations (the *path*) can then be given as

$$X_T = \{x_1, x_2, ...x_T\}$$

where T is the terminal time. The initial position x_0 is known, while the others have to be estimated.

Odometry is the most common method used to obtain the relative information between two consecutive locations. For wheeled robots, this can be obtained from the robot's wheel encoders or from the controls given to its motors. Additional sensors and techniques will be required for obtaining odometry information in non-wheeled robots. A visual odometry algorithm for estimating the trajectory of an aerial robot using just an on-board monocular camera had been developed as part of the second-term mini-project (Visual Odometry and PID Controller for an Aerial Robot).

Let u_t be the motion data that characterizes the motion between time $t-1$ and time t. Now the relative motions of the robot can be given by the following sequence:

$$U_T = \{u_1, u_2, u_3, ...x_T\}$$

(Note that u_1 defines the motion from the initial position x_0 to x_1.)

Ideally, the odometry would be noise-free and U_T would be sufficient to recover the past X_t from the initial location x_0. However, odometry measurements are noisy and also extremely sensitive to errors. Therefore, path integration techniques inevitably diverge over time.

It is worth noting that the motion data u here was assumed to be the odometry information. In practice however, two different models of motion data can be used. The first model, as discussed, assumes that the motion data u_t specifies the odometry information. Most commercial bases provide odometry using the kinematic information (distance travelled, angle turned etc.). The second model assumes that the motion data u_t specifies the velocity commands given to the robots motors. Many commercial mobile robots (e.g., differential drive and synchro-drive) can be considered to be actuated by independent translational and rotational velocities with odometry information.

In practice however, it is seen that odometry models are more accurate than velocity models, as most commercial robots do not execute velocity commands with the level of accuracy that can be obtained by measuring the revolution of the robots wheels [11]. However, odometry is only available once the motion is executed. Hence it cannot be used for motion planning. Planning algorithms such as collision avoidance have to predict the effects of motion. Thus, odometry models are usually applied for estimation, whereas velocity models are used for probabilistic motion planning. Since the project involves only state estimation and not motion planning, the odometry model (the visual odometry algorithm from the mini-project) was adopted for this project, and further discussions in the report does not focus on the velocity model.

Finally, the robot senses the objects in the environment using some sensor. Let m denote the map of the environment. It describes the locations of the distinguishable landmarks in the environment. Typically, the environment map m is assumed to be static, i.e., consists of stationary objects only.

The measurements obtained from the sensors establish information relating the features in m and the robot location x_t. Without loss of generality and assuming that the robot takes one measurement at each point in time, the measurements can be given by the sequence

$$Z_T = \{z_1, z_2, z_3, ...z_T\}$$

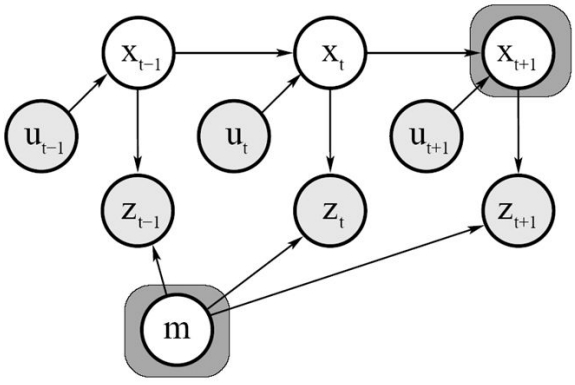


Figure 2.2: Dynamic Bayesian Network Model of SLAM Process
Source: [9]

Figure 2.2 shows a graphical model of the SLAM process. Such a model is called the Dynamic Bayesian Network (DBN) model of the SLAM problem. A Bayesian network is a graphical model that describes a stochastic process as a directed graph [1]. The graph has one node for each random variable in the process. The arcs indicate the causal relationships, and the shaded nodes are variables that are directly observable to the robot. The SLAM problem is now the problem of recovering the unobservable variables in the model, i.e the model of the world m and the sequence of robot locations X_T.

Filtering and Smoothing Approaches

The literature classifies the SLAM approaches into two main types [9]: filtering and smoothing. Filtering approaches model SLAM as an online state estimation problem of estimating the posterior over the *momentary* pose along with the map. The estimate is augmented and refined by incorporating the new measurements as soon as they become available. Hence it is also called the *online* SLAM problem. Online SLAM seeks to recover the present robot location, and not the entire path. Popular techniques such as Kalman filter, information filters and particle filters fall in this category [12, 13, 14, 15]. For online SLAM, the posterior, i.e., the probability of the robot being at location x_m in the map m after making the observations given by Z_T and odometry sequence U_T, can be given by

$$p(x_t, m \,|\, Z_T, U_T, x_0) \tag{2.1}$$

Smoothing approaches, on the other hand, estimate the full trajectory of the robot from the full set of measurements. Instead of using just the current pose of the robot, the entire path along with the map is used for calculating the posterior. These approaches address the so-called *full* (or *offline*) SLAM problem, and they typically rely on least-square error minimization techniques. Various pose-graph-based SLAM methods fall in to this category [16, 17, 18, 19, 20]. The offline SLAM problem is defined via

$$p(X_T, m \,|\, Z_T, U_T, x_0) \tag{2.2}$$

Written in this way, the full SLAM problem is clearly the problem of calculating the joint posterior probability over X_T and m from the available data. It can be seen that in both equations (2.1 and 2.2), all the variables on the right side of the conditioning bar are directly observable to the robot, whereas those on the left have to be estimated.

To solve either SLAM problem, the robot needs to be provided with two more things: the measurement model and the motion model [9].

The Measurement Model

The modelling of the measurement setup begins with the *exact*, noise-free measurement function. The measurement function h describes the sensors of the robot. It should accept as input a description of the environment m and the location of the robot x_t, and should compute the measurement:

$$h(x_t, m)$$

Since the measurements are inherently noisy, the probabilistic measurement model (or the observation/sensor model) is derived from this measurement function by adding a noise term. The measurement model can now be considered to be a probability distribution function with its peak at the noise-free value of $h(x_t, m)$, that also allows for measurement noise, given as:

$$p(z_t \,|\, x_t, m) \sim \mathcal{N}(h(x_t, m), Q_t)$$

where $\mathcal{N}$ denotes the normal distribution centred at $h(x_t, m)$, and Q_t is the covariance matrix at time t.

The observation model thus denotes the probability of performing the observation z_t given that the robot is at location x_t in the map m. It is represented by the arrows entering z_t in Figure 2.2. It can be seen that the exteroceptive observation z_t depends only on the current location x_t of the robot, and the map m.

The Motion Model

The motion (or transition/action) model can be derived from the kinematic model of the robot motion. If the location vector x_{t-1} and the motion u_t is known, basic kinematics can be used to calculate x_t. For an ideal odometry model, it could be as simple as $x_{t-1} + u_t$. However, in general, let this kinematic function be g:

$$g(x_{t-1}, u_t)$$

The motion model is then defined by a normal distribution centred at $g(x_{t-1}, u_t)$, and subject to Gaussian noise:

$$p(x_t \mid x_{t-1}, u_t) = \mathcal{N}(g(x_{t-1}, u_t),\, R_t)$$

where R_t is the covariance matrix.

The motion model plays the role of the state transition model in mobile robotics. It is represented by the two edges leading to x_t in Figure 2.2, and describes the posterior distribution over the kinematic states that the robot assumes when executing the odometry u_t when its pose is x_{t-1}.

Landmark Selection and the Data Association Problem

A definite representation of the environment in the form of a map is essential for developing any SLAM algorithm, and landmark-based representations are the most widely used due to ease of representation and matching. Dense representations are usually used in conjunction with range sensors [14].

A landmark is a distinct feature in the environment that a robot can recognize from its sensory inputs. They can be basic geometric shapes such as lines and rectangles, or distinguishable point features such as corners. They could also be special markers which may include additional information to guide the robot (e.g. in the form bar-codes or QR codes).

Deciding what features should be chosen as landmarks is critical in SLAM. They should be distinguishable features in the world which can be easily re-observed. They should be stationary in the world and unique so that they can be identified from one time-step to another without being mixed up with other landmarks. Ideal landmarks are re-observable from multiple views, allowing them to be detected from different positions and thus from different angles.

Two types of landmarks are mentioned in literature [21]: *artificial* and *natural*. Natural landmarks are those objects or features that are already in environment, and have not been placed for enabling navigation. Conversely, artificial landmarks are specially designed objects (markers) that are placed with the sole purpose of enabling the navigation of the robot. Their detection is much easier as they are designed for optimal contrast [22]. Additionally, the identity, shape, and size of these landmarks are known in advance.

Since using artificial landmarks is not applicable in real-life situations without setting up the environment beforehand, modern SLAM researchers mainly focus on building maps from the natural landmarks in a scene. The main problem in natural landmark navigation is detecting and matching characteristic features from the sensory inputs. The selection of the landmarks is important since it will determine the complexity in its description, detection, and matching. Proper selection of features will also reduce the chances for ambiguity and increase positioning accuracy.

Once the landmarks have been detected and described, the core problem, and possibly the most challenging problem in SLAM, is associating them with previous observations.

This is the problem of data association - the task of placing measurements into one of the following categories [23]:

- Associated with as yet unknown regions of environment (new landmarks).
- Associated with already known (mapped) region.
- Not enough evidence to make a decision (association left pending)
- Spurious (noisy/false measurement)

Since the steps following landmark detection in SLAM depend hugely on which of the above categories the landmarks are placed, a good method of associating them with the current map is necessary for building an accurate map of the environment. Associating the measurements of the same landmarks in subsequent time-frames is necessary for proper localisation and mapping, and also for estimating the positions of new landmarks relative to them. Hence, when landmarks are observed, basic SLAM algorithms try to find their correspondences from the last time-frame, or from a set of observations made in the last few time-frames.

While in literature, point-landmark problems using range-bearing sensors are by far the most studied [9], SLAM algorithms are not confined to landmark worlds or range-bearing sensors. As long as there is crisp definition of the features in m, the only other requirements for developing a SLAM algorithm are the measurement model and the motion model.

Loop Closure

Drift in the state estimation and errors in mapping over time are inevitable and has to be expected. This can be attributed to the number, type and quality of sensors, and various other sources of noise in the system. The errors accumulate over time, consequently increasing the uncertainty of beliefs as the algorithm progresses. Therefore, the most recent estimations will tend to deviate more from the ground truth than the earlier estimations.

However, if the robot observes a landmark whose position in the world is known (an artificial landmark), it can correct its pose and/or the map of the world, relative to the position of the landmark. This would give the robot the evidence required to update its beliefs and correct its current, and even some of its recent previous state estimations. A typical SLAM problem, however, aims to build a map without any prior knowledge of the landmarks in the world (i.e., using only natural landmarks). So the robot has to rely on the map that it has built so far. A loop closure occurs when the robot observes a landmark that it had previously come across, and successfully identifies it. Due to the accumulating drift and map estimation errors, the earlier estimate of the landmark position will be more reliable than its current estimate. Therefore, loop closures can be used to update the beliefs of the system, thereby improving the state estimations of the robot and/or the generated map.

Loop closure detection is the problem of recognizing a location that had been previously visited. This essentially makes loop closure detection a data association problem of matching the current observation to an observation made earlier in time. However, in this case, the search space is much larger than just the previous few time-frames. Typical loop closure methods apply a second algorithm to measure some type of sensor similarity, and reset the location priors when a match is detected.

Depending on the type of sensor used for observations, different approaches have been developed for loop closure detection. Scan-to-scan matching techniques are often used in laser-based SLAM, where the scanner readings at each time-step is compared with

the scans of the previous time-steps in the map memory [24, 25]. Scan-to-map matching techniques, where the best match for the current scan is searched for in the map as a whole rather than individual frames, have been shown to have better loop closure detection [26]. Loop closure detection in visual SLAM systems have been implemented using image-to-image matching [27, 28, 29], image-to-map [30], and map-to-map [31] techniques (Section 4.3). Robust loop closure detection using a combination of laser and camera information has been shown in [23]. Since the project aims to develop a SLAM system for a robot having only a monocular camera for sensory inputs, this report focusses on vision-based loop closure detection techniques.

Chapter 3

Graph-Based SLAM

An intuitive way of addressing the SLAM problem is via its so called graph-based formulation which highlights its underlying spatial structure (Figure 3.1). In graph-based approaches, the entire model is represented as a graph whose nodes correspond to robot poses or landmarks, and edges encode a sensor measurement that constrains the nodes [1, 25]. These measurement constraints are obtained from observations of the environment or from movement actions taken by the robot. The measurements are obviously not 'hard' constraints and can be contradictory since observations are always affected by noise. Once such a graph is created, graph-based SLAM algorithms try to find a configuration of nodes that would best fit the measurements made. This involves solving a large error minimisation problem. Since graph-based SLAM techniques use the *full* set of measurements to estimate the trajectory, it falls under the *smoothing/offline* category of SLAM approaches. Thus the graph-based SLAM problem consists of two separate tasks:

(i) *Graph construction*: constructing the graph from the raw measurements, and

(ii) *Graph optimization*: determining the most likely configuration of the poses given the edges of the graph.

The graph construction is usually called the *front-end* of graph-based SLAM, which is responsible for adding nodes and constraints to the graph, while the graph optimization is called its *back-end* [1], which attempts to find the optimum configuration of the graph. The back-end does not directly depend on the sensors of the system and is therefore sensor agnostic, whereas the front-end relies heavily on the sensors as the type of measurement data obtained will define the method of transforming this data into the graph, which then has to be passed to the back-end.

Most attempts at the graph-based approaches in literature focus on devising efficient implementations of back-end optimisation techniques, assuming a consistent graph was already constructed by the front-end. The Sparse Least Squares on Manifolds (SLoM) back-end [32] and the General (Hyper) Graph Optimization (g^2o framework) [33] framework are implementations of the graph optimisation for smooth manifolds (Appendix A.3) taking advantage of the sparse structures in the problem (Section 3.3.2). In [34], the author approaches the optimisation problem using stochastic gradient descent for improving estimations for larger trajectories. An extension of this method using a tree parametrisation to improve the convergence speed of the algorithm is presented in [35].

The advantage of these smoothing/offline approaches is that the optimisation can be revised and rectified, since the full set of data is available throughout the whole mapping process. Additionally, the computational complexity of these approaches grows only linearly on the size N of the trajectory, in contrast to the quadratic (for Extended Kalman Filter and its variants) or cubic complexity increase (for information filter) of the filtering

approaches [36]. Another advantage over the filtering methods is that the information matrix associated with smoothing remains sparse (Section 3.3.2), whereas the Kalman filter covariance matrix (or the information matrix associated with the information filter) become fully dense over time. This sparse matrix can be factorized efficiently using sparse Cholesky (Appendix A.1) or QR factorization, yielding a square root information matrix that immediately yields the optimal robot trajectory and map [17].

Since smoothing methods typically store the entire trajectory in memory, a disadvantage of the approach is that the computational complexity grows unbound over time. In most typical mapping scenarios, however, the EKF covariance matrix will grow much faster. Also, as with all information matrix approaches, it is hard to recover the covariance matrix governing the unknowns [17]. Another problem that arises in the implementation of the graph-based SLAM problem is choosing the information matrices for the measurements that would define the constraints of the graph. No closed-form solutions are generally available, and the information matrices are usually defined empirically according to the algorithm, type of sensors and measurements, and the environment to be mapped [16, 37, 4].

Graphical Representation of SLAM

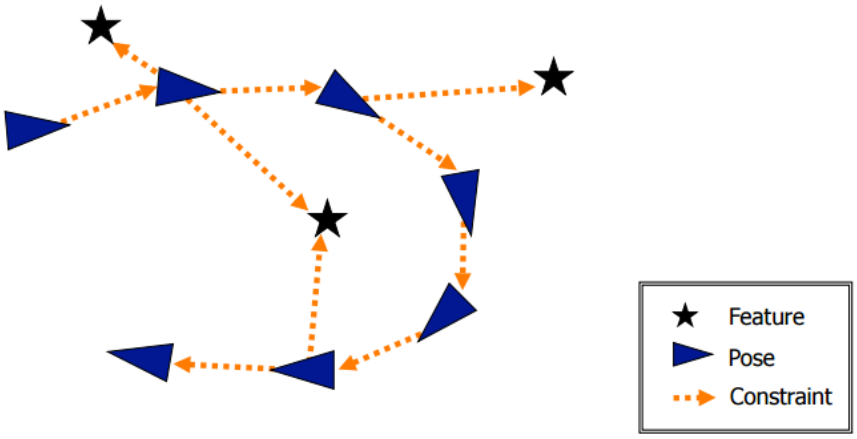


Figure 3.1: Graphical Representation of SLAM Process
Source: Jinyong, 'Graph-based SLAM with Landmark'
Retrieved from: `http://jinyongjeong.github.io/2017/02/26/lec14_Least_square_SLAM_landmark/`

Graph-based representation is an alternative to the DBN model (Figure 2.2) of the SLAM problem, created out of raw sensor measurements. Each node in the graph (Figure 3.1) represents a landmark position or a robot pose and a measurement obtained at that pose. The edges of the graph encodes the spatial constraints between the nodes that result from observations z_t or from odometry measurements u_t. Once the graph is constructed, the algorithm seeks to find the configuration of nodes that best satisfies the constraints.

In graph-based SLAM, the raw sensor measurements, as mentioned previously, are replaced by the edges in the graph which can then be seen as "virtual" measurements. These edges (or constraints) between two nodes are probability distributions over the relative transformations between the two nodes. These transformations are either odometry measurements between subsequent robot poses or are calculated by aligning the observations acquired at the two robot poses.

However, a single observation z_t might result in multiple potential edges connecting

different poses in the graph. Therefore the observation model $p(z_t \mid x_t, m)$ is multi-modal and the Gaussian assumption does not hold. Directly dealing with this multi-modality in the estimation process would lead to an exponentially-increasing complexity. Therefore, most practical approaches restrict the estimate to the most likely topography, which means that the most likely constraint resulting from the observation has to be determined [1]. This is part of the data association problem discussed in Section 2.2.

If the observations are locally affected by Gaussian noise and the data association is known, the goal of a graph-based mapping algorithm becomes the computation of a Gaussian approximation of the posterior over the robot trajectory, such that the final configuration of the nodes (X_T and m) maximises the likelihood of the observations Z_T. Therefore, the graph-based SLAM approach is in its core a constraint optimisation problem.

The problem of graph-based SLAM is almost always approached with the least squares methodology, so much so that graph-based SLAM is often referred to as 'least-squares SLAM' [38, 39]. The following sections would therefore explain how to approach a state estimation problem using least squares and Gauss-Newton error minimisation, before moving to the constraint optimisation problem that is graph-based SLAM. It has to be noted that this chapter is meant to be a basic tutorial to the formulation and methodology involved in solving the graph-based SLAM problem, and has overlooked many practical and technical issues that can arise while performing its constraint optimisation (a few are discussed in Appendix A).

Mathematical Basis - State Estimation Using Least Squares

Least squares is a standard approach to a large set of error minimisation problems, developed by Carl Friedrich Gauss in 1795 [38]. It is a technique of solving an overdetermined system which has 'more equations than unknowns'. It obtains the best solution by minimising *the sum of the squared errors* in the equations. The following subsections will attempt to explain the methodology involved in applying the least squares technique to the problem of state estimation in a non-linear stationary system.

The Non-Linear System Setup

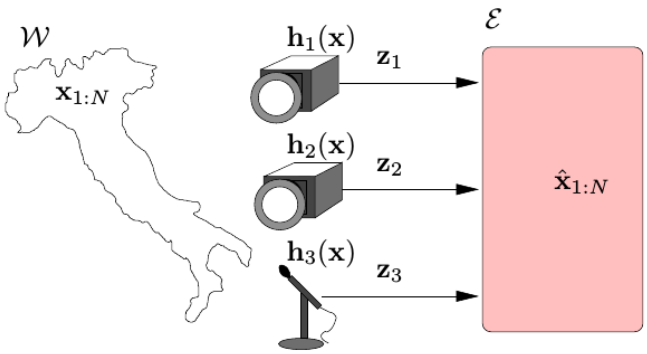


Figure 3.2: Illustration of a State Estimation on a Stationary System

$\mathcal{W}$ is the phenomena being observed. The state of this phenomena is indirectly measured through a set of sensors, each of which measures a quantity z_k that depends on the state of the system.

Source: [39]

Consider a stationary system $\mathcal{W}$, whose state is parametrized by a set of state vari-

ables: $\mathbf{x} = \{x_1, x_2, ..., x_N\}$. These variables can span over arbitrary spaces. However, for mathematical simplicity, they will be assumed to be in Euclidean space[1]. The state of the system can be can be observed indirectly using a set of sensors, giving a set of measurements $\mathbf{z} = \{z_1, z_2, ..., z_k\}$. Measurements are affected by noise, which means z_k are in fact random variables. The setup is illustrated in Figure 3.2.

The measurements received by the sensors depend on the state of the system, and it is assumed that the state models all the knowledge required to predict the distribution of the measurement, i.e., the sensor model is defined.

The requirement is to find the state vector $\mathbf{x}$ from the measurements. However, due to noisy measurements, it is not possible to estimate the *exact* state of the system given the measurements. Instead, a distribution over the potential states of the system given these measurements can be computed. Formally, the posterior to be obtained is

$$p(\mathbf{x} \,|\, \mathbf{z}) = p(x_1, ..., x_N \,|z_1, ..., z_K) = p(x_{1:N}|z_{1:K}) \tag{3.1}$$

Complexity in Direct Estimation of the Posterior

However, there are in general no closed formed solutions to obtain Equation 3.1 [39], for reasons including:

- The measurements z_k is generally obtained from observing only a subset of the state variables.
- The mapping between the measurements and the states (action model and measurement model) are generally non-linear, making the probability distribution multimodal and complex.
- The number of measurements at each state may be arbitrary. Therefore, sometimes there might not be enough measurements to characterize the distribution over states, while sometimes the measurements may not be meaningful for any particular state due to noise.
- Wrong measurements may be present.

However, there is another distribution that is easier to obtain: the sensor model, $p(z_k|\mathbf{x})$. This is a predictive distribution that tells the probability of obtaining a measurement z_k, assuming that the state $\mathbf{x}$ is known. Another important feature of the measurement model is that the measurements are independent of each other, given the state. In probabilistic form, this independence assumption would imply:

$$p(z_{1:K}|x_{1:N}) = \prod_{k=1}^{K} p(z_k|x_{1:N}) \tag{3.2}$$

The term $p(z_{1:K}|x_{1:N})$ is known as the *likelihood* of the measurements given the states. This information can then be used to obtain the required posterior (Equation 3.1), with the help of the Bayes' rule, highlighting the contribution of the likelihood:

$$p(x_{1:N}|z_{1:K}) = \frac{\overbrace{p(z_{1:K}|x_{1:N})}^{likelihood} \cdot \overbrace{p(x_{1:N})}^{prior}}{\underbrace{p(z_{1:K})}_{normalizer}}$$

[1]State variables are generally considered to be in the space of manifolds, since they also contain robot orientations. Rotations are not Euclidean, and are smooth manifolds. Refer Appendix A.3.

In the above equation, the prior $p(x_{1:N})$ models the potential knowledge about the states that is known before taking any measurements. If nothing is known in advance, the prior is a uniform distribution whose value is a constant p_x. Since all the measurements are known and the normalizer $p(z_{1:K})$ is independent of the states, the normalizer becomes another constant number p_z. So the above equation can be rewritten as

$$p(x_{1:N}|z_{1:K}) = \frac{p(z_{1:K}|x_{1:N})p_x}{p_z}$$

To drop the fraction, a non-negative constant η can be defined such that $\eta = 1/p_z$. The equation now becomes

$$p(x_{1:N}|z_{1:K}) = \eta p_x p(z_{1:K}|x_{1:N})$$

Using Equation 3.2 in the above equation, and removing the constants,

$$p(x_{1:N}|z_{1:K}) \propto \prod_{k=1}^{K} p(z_k|x_{1:N}) \tag{3.3}$$

Gauss-Newton Error Minimisation

As evident from Equation 3.3, the distribution over the possible states given the measurements is proportional to the likelihood of the measurement given the states. As is standard in SLAM problems, the measurement noise can be assumed to be Gaussian. If the noise affecting the measurements is normally distributed with zero mean, the likelihood of the measurement will also be Gaussian, giving

$$p(z_k|\mathbf{x}) \propto \exp\big(-(\hat{z}_k - z_k)^T \Omega_k (\hat{z}_k - z_k)\big) \tag{3.4}$$

Here $\Omega_k = \Sigma_k^{-1}$ (inverse of the covariance matrix) is the information matrix of the conditional measurement, while $\hat{z}_k$ is the predicted measurement. The predicted measurement represents the mean of the conditional distribution. Similarly, being the product of Gaussians (according to Equation 3.2), the likelihood of all the measurements $p(\mathbf{z}|\mathbf{x})$ will also be Gaussian.

The predicted measurement z_k is obtained by applying the sensor model $h_k(\cdot)$ over the states,

$$\hat{z}_k = h_k(\mathbf{x})$$

If the sensor model had been a linear mapping between the state and the measurement, such as an affine transformation $h_k(\mathbf{x}) = A\mathbf{x} + b$, and the noise in the measurements was Gaussian, then $p(\mathbf{x}|\mathbf{z})$ would have been normally distributed $\mathcal{N}(\mu_\mathbf{x}, \Sigma_\mathbf{x})$ [40].

Unfortunately, the sensor model is in general a non-linear function of the states. However, if the function is smooth enough, it can be linearised locally by approximating it in the neighbourhood of a linearisation point $\breve{\mathbf{x}}$ by its Taylor expansion

$$h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) \simeq \underbrace{h_k(\breve{\mathbf{x}})}_{\breve{z}_k} + \underbrace{\left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{J}_k} \cdot \Delta\mathbf{x} \tag{3.5}$$

where $\mathbf{J}_k$ is the Jacobian of $h_k(\mathbf{x})$.

If state $\mathbf{x}^*$ is the linearisation point that better explains the measurements, it can be obtained by maximising the measurement likelihood:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} p(\mathbf{z}|\mathbf{x}) \tag{3.6}$$

By plugging in Equation 3.5 in Equation 3.4, a new distribution over $p(\mathbf{z}|\Delta\mathbf{x})$ governed by the increments $\Delta\mathbf{x}$ can be obtained:

$$p(z_k|\mathbf{x}^* + \Delta\mathbf{x}) \propto \exp\Big(-\big(h_k(\mathbf{x}^*) + \mathbf{J}_k\Delta\mathbf{x} - z_k\big)^T \Omega_k \big(h_k(\mathbf{x}^*) + \mathbf{J}_k\Delta\mathbf{x} - z_k\big)\Big) \tag{3.7}$$

Evaluating Equation 3.7 at different values of $\Delta\mathbf{x}$ can tell us how the likelihood fits the data measurement given a perturbation of the optimal state. Since $\mathbf{x}^*$ gives the optimal state given the measurements, it can be treated as a constant. However, as mentioned before, the sensor model is non-linear, and therefore finding the state $\mathbf{x}^*$ requires some work.

The state $\mathbf{x}^*$ should be such that it maximises Equation 3.4

$$\mathbf{x}^* = \arg\max \prod_{k=1}^{K} p(z_k|\mathbf{x}) \tag{3.8}$$

By the Gaussian assumption (Equation 3.4), this becomes

$$\mathbf{x}^* = \arg\max \prod_{k=1}^{K} \exp[-(\hat{z}_k - z_k)^T \Omega_k (\hat{z}_k - z_k)]$$

For easier calculations, the exponent function can be removed by taking the logarithm of the equation:

$$\mathbf{x}^* = \arg\max \sum_{k=1}^{K} [-(\hat{z}_k - z_k)^T \Omega_k (\hat{z}_k - z_k)] \tag{3.9}$$

Removing the minus sign, the equation becomes

$$\begin{aligned} \mathbf{x}^* &= \arg\min \sum_{k=1}^{K} (\hat{z}_k - z_k)^T \Omega_k (\hat{z}_k - z_k) \\ &= \arg\min \sum_{k=1}^{K} \big(h_k(\mathbf{x}) - z_k\big)^T \Omega_k \big(h_k(\mathbf{x}) - z_k\big) \end{aligned} \tag{3.10}$$

A new error function $e_k(\mathbf{x})$ can now be introduced into the equation, by defining it as

$$e_k(\mathbf{x}) = \hat{z}_k - z_k = h_k(\mathbf{x}) - z_k \tag{3.11}$$

By plugging this error term into Equation 3.10, the final argument of minimisation is obtained as:

$$\boldsymbol{F}(\mathbf{x}) = \arg\min \sum_{k=1}^{K} \underbrace{e_k(\mathbf{x})^T \Omega_k e_k(\mathbf{x})}_{\mathbf{e}_k(\mathbf{x})} \tag{3.12}$$

It is interesting to note here that minimising the squared error (Equation 3.12) is equivalent to maximising the likelihood of the independent Gaussian distributions of the measurement model (Equation 3.8).

Assuming that a reasonably good estimation of the optimum $\breve{\mathbf{x}}$ is available, one of the summands in Equation 3.10 can be rewritten by exploiting the Taylor approximation (Equation 3.5) as follows:

$$
\begin{aligned}
\mathbf{e}_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) &= \left(h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) - z_k\right)^T \Omega \left(h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) - z_k\right) \\
&\quad [\text{Plugging in Equation 3.5 (Taylor approximation)}] \\
&= \left(\mathbf{J}_k\Delta\mathbf{x} + h_k(\breve{\mathbf{x}}) - \hat{z}_k\right)^T \Omega \left(\mathbf{J}_k\Delta\mathbf{x} + h_k(\breve{\mathbf{x}}) - \hat{z}_k\right) \\
&\quad [\text{Substituting } h_k(\breve{\mathbf{x}}) - z_k = \mathbf{e}_k] \\
&= \left(\mathbf{J}_k\Delta\mathbf{x} + \mathbf{e}_k\right)^T \Omega \left(\mathbf{J}_k\Delta\mathbf{x} + \mathbf{e}_k\right) \\
&= \Delta\mathbf{x}^T \underbrace{\mathbf{J}_k\Omega_k\mathbf{J}_k}_{\mathbf{H}_k} \Delta\mathbf{x} + 2\underbrace{\mathbf{e}_k^T\Omega_k\mathbf{J}_k}_{\mathbf{b}_k^T}\Delta\mathbf{x} + \underbrace{\mathbf{e}_k^T\Omega_k\mathbf{e}_k}_{c_k} \qquad (3.13) \\
&= \Delta\mathbf{x}^T\mathbf{H}_k\Delta\mathbf{x} + 2\mathbf{b}_k^T\Delta\mathbf{x} + c_k \qquad (3.14)
\end{aligned}
$$

Substituting Equation 3.14 in Equation 3.12

$$
\begin{aligned}
\boldsymbol{F}(\breve{\mathbf{x}} + \Delta\mathbf{x}) &\simeq \sum_{k=1}^{K} \Delta\mathbf{x}^T\mathbf{H}_k\Delta\mathbf{x} + 2\mathbf{b}_k^T\Delta\mathbf{x} + c_k \\
&= \Delta\mathbf{x}^T \underbrace{\left[\sum_{k=1}^{K}\mathbf{H}_k\right]}_{\mathbf{H}}\Delta\mathbf{x} + 2\underbrace{\left[\sum_{k=1}^{K}\mathbf{b}_k^T\right]}_{\mathbf{b}^T}\Delta\mathbf{x} + \underbrace{\sum_{k=1}^{K}c_k}_{c} \qquad (3.15) \\
&= \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + 2\mathbf{b}^T\Delta\mathbf{x} + \mathbf{c} \qquad (3.16)
\end{aligned}
$$

By doing this, we get a quadratic expression of the objective function, under a linear approximation of the sensor model around a neighbourhood of the initial estimate $\breve{\mathbf{x}}$. This means that if the initial estimate is fixed, the value of the function in the neighbourhood can be approximated by a quadratic form with respect to the increment $\Delta\mathbf{x}$. The increment $\Delta\mathbf{x}$ – which when applied to the current guess would yield a better solution – can be obtained by minimising this quadratic form.

This can be calculated by setting the derivative of the function to zero, and solving the linear equation.

$$
\begin{aligned}
\frac{\partial}{\partial\Delta\mathbf{x}}\left(\Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + 2\mathbf{b}^T\Delta\mathbf{x} + c\right) &= 2\mathbf{H}\Delta\mathbf{x} + 2\mathbf{b} = 0 \\
\Rightarrow \mathbf{H}\Delta\mathbf{x} &= -\mathbf{b} \qquad (3.17)
\end{aligned}
$$

This increment is then added to the estimate $\breve{\mathbf{x}}$

$$\breve{\mathbf{x}} = \breve{\mathbf{x}} + \Delta\mathbf{x}$$

As mentioned earlier, if the measurement function was linear, the solution for the minimisation would be straightforward and will be attained in one iteration of these steps. Then the optimum value of states $\mathbf{x}^*$ would be equal to the new value of $\breve{\mathbf{x}}$. However, this is typically not the case, and the procedure has to be repeated by locally linearising the error function using Taylor approximation at each iteration until no significant improvements in the value of $\mathbf{x}$ are observed. The whole procedure is called the Gauss-Newton error minimisation and is formally described in Algorithm 1. In practice however, the state estimation problem is usually solved using the Levenberg-Marquardt error minimisation algorithm due to its 'error-recovery' nature that allows it to correct itself when an increment does not decrease the error in the system (Appendix A.2).

Algorithm 1 Gauss Newton Error Minimisation Algorithm

Require: Initial Guess $\breve{\mathbf{x}}$; and measurements $\mathcal{C} = \{\langle z_k(\cdot), \Omega_k \rangle\}$
Ensure: $\mathbf{x}^*$ = new solution

$\boldsymbol{F}_{new} \leftarrow \breve{\boldsymbol{F}}$ ▷ *current error*

// iterate till no substantial improvements attained

repeat

  $\breve{\boldsymbol{F}} \leftarrow \boldsymbol{F}_{new}$

  $\mathbf{b} \leftarrow 0 \quad \mathbf{H} \leftarrow 0$

  for all $k = 1...K$ **do**

    // compute prediction and error

    $\breve{z}_k \leftarrow h_k(\breve{\mathbf{x}})$

    $\mathbf{e}_k \leftarrow \hat{z}_k - z_k$

    $\mathbf{J}_k \leftarrow \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$

    // computing contribution of the measurement in the linear system

    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{J}_k; \quad \mathbf{b}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{e}_k$

    // accumulating contributions to build overall system

    $\mathbf{H}$ += $\mathbf{H}_k$; $\mathbf{b}$ += $\mathbf{b}_k$

  end for

  // solve linear equation to get new increment

  $\Delta\mathbf{x} \leftarrow \text{solve}(\mathbf{H}\Delta\mathbf{x} = -\mathbf{b})$

  // update state

  $\breve{\mathbf{x}}$ += $\Delta\mathbf{x}$

  $\boldsymbol{F}_{new} \leftarrow \boldsymbol{F}(\breve{\mathbf{x}})$ ▷ *new error*

until $\breve{\boldsymbol{F}} - \boldsymbol{F}_{new} < \epsilon$

return $\breve{\mathbf{x}}$

Mathematical Formulation of Graph-Based SLAM

Since graph-based methods try to obtain a solution from the full set of robot poses and observations, they belong to the *full* (or *offline*) approach to the SLAM problem, and is therefore defined by Equation 2.2:

$$p(X_T, m \,|\, Z_T, U_T, x_0)$$

The initial position of the robot (x_0) is assumed to be fixed (known or chosen arbitrarily). The poses in X_T and the odometry data in U_T are usually represented as 2D or 3D transformations in SE(2) or in SE(3). The map will be assumed to be represented using landmarks.

This section will explain the formulation of the graph using vector notations and solving the constraint optimisation problem using the least-squares approach explained previously.

Graph Optimisation Using Least-Squares

The graph to be constructed will have nodes that represent the poses attained by the robot and the corresponding measurement taken at each time-frame during mapping. Every edge between two nodes will correspond to a spacial constraint between them. Once the best configuration of nodes explaining the observations is obtained, the map can be rendered using the known poses and the corresponding measurements.

Let $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_T\}$ be a vector of parameters, where $\mathbf{x}_i$ describes the pose (a 2D or 3D transformation matrix) of the node i. Edges can be formed in two ways: (i) When the robot moves from x_i to x_{i+1}, corresponding to the odometry u_{i+1}, or (ii) when the robot observes the same part of the environment from x_i and x_j, which are far apart in time. This creates a *virtual measurement* of the pose of x_j as observed from x_i.

Let $\mathbf{z}_{ij}$ and Ω_{ij} be the mean and the information matrix of a virtual measurement (or odometry measurement) respectively, between the nodes i and j. This virtual measurement is a transformation that makes the observations acquired from i maximally overlap with the observations from j. Let $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ be the prediction of a virtual measurement, given a configuration of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Usually this prediction is the relative transformation between the two node poses. So the error term required for the optimisation (Figure 3.3) can be defined using Equation 3.11 as:

$$e_{ij} = \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}_{ij} \tag{3.18}$$

As seen in Figure 3.3, an edge originates from the measurement $\mathbf{z}_{ij}$. If the relative position of the two nodes (given by the transformation $\mathbf{x}_i^{-1}\mathbf{x}_j$) is available, it is possible to compute the expected measurement $\hat{\mathbf{z}}_{ij}$ that represents $\mathbf{x}_j$ as seen from the frame of $\mathbf{x}_i$. The error $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ depends on the displacement between the expected and the real measurement. An edge is fully characterized by its error function $\mathbf{e}_{ij}$ and by the information matrix Ω_{ij} of the measurement that accounts for its uncertainty.

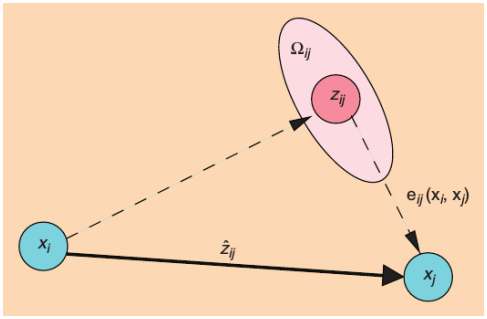


Figure 3.3: Illustration of Edge Formation Between Two Nodes

Source: [1]

If $\mathcal{C}$ is the set of pairs of indices for which a constraint $\mathbf{z}$ exists, the minimisation argument (according to Equation 3.12) becomes:

$$\boldsymbol{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{e_{ij}^T \Omega_{ij} e_{ij}}_{\mathbf{e}_{ij}(\mathbf{x})} \tag{3.19}$$

Clearly, minimising $\boldsymbol{F}(\mathbf{x})$ would give the optimal configuration of the nodes, $\mathbf{x}^*$.

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \boldsymbol{F}(\mathbf{x}) \tag{3.20}$$

Now that the minimisation argument is defined, the graph optimisation problem can be solved using the Gauss-Newton (Algorithm 1) or Levenberg-Marquardt (Algorithm 4) algorithms, by assuming a good initial guess $\breve{\mathbf{x}}$ and linearising the error function around the current guess using the Taylor approximation (as explained in Section 3.2.3).

Accordingly, the error function approximation at each constraint will be:

$$\mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \simeq (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x})^T \Omega (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x}) \tag{3.21}$$

$$= \Delta\mathbf{x}^T \underbrace{\mathbf{J}_{ij}\Omega_{ij}\mathbf{J}_{ij}}_{\mathbf{H}_{ij}} \Delta\mathbf{x} + 2\underbrace{\mathbf{e}_{ij}^T \Omega_{ij}\mathbf{J}_{ij}}_{\mathbf{b}_{ij}^T} \Delta\mathbf{x} + \underbrace{\mathbf{e}_{ij}^T \Omega_{ij}\mathbf{e}_{ij}}_{c_{ij}} \tag{3.22}$$

Following the same steps from the previous state estimation problem with this local approximation, the function $\boldsymbol{F}(\mathbf{x})$ can be rewritten as

$$\boldsymbol{F}(\mathbf{x}) = \sum_{\langle i,j\rangle \in \mathcal{C}} \mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \tag{3.23}$$

$$\simeq \sum_{\langle i,j\rangle \in \mathcal{C}} \Delta\mathbf{x}^T \mathbf{H}_{ij} \Delta\mathbf{x} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + c_{ij} \tag{3.24}$$

$$= \Delta\mathbf{x}^T \mathbf{H} \Delta\mathbf{x} + 2\mathbf{b}^T \Delta\mathbf{x} + \mathbf{c} \tag{3.25}$$

After taking the derivative and equating to zero, the required increment $\Delta\mathbf{x}$ can be obtained by solving

$$\mathbf{H}\Delta\mathbf{x} = -\mathbf{b} \tag{3.26}$$

By using memory efficient representations and calculation techniques such as the sparse Cholesky factorisation (Appendix A.1), this linear equation can be solved efficiently to obtain the required increment $\Delta\mathbf{x}$ at each iteration [38, 39, 1]. The linearised solution at the iteration is then obtained by adding to the current guess the computed increments

$$\mathbf{x}^* = \breve{\mathbf{x}} + \Delta\mathbf{x}$$

Structural Considerations in the Optimisation Problem

An important feature of this linearised system that improves its calculation efficiency is the sparse structures of its matrices. According to Equations 3.25 and 3.22, the matrix $\mathbf{H}$ and the vector $\mathbf{b}$ are obtained by summing up a set of matrices and vectors, one for each constraint. Each constraint will contribute to the system with an addend term. The structure of this addend depends on the Jacobian of the error function. Clearly, each error function corresponding to a constraint depends only on the values of the two relating nodes. Therefore, the Jacobian in Equation 3.21 has the following form:

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0} & \cdots & \mathbf{0} & \underbrace{\mathbf{A}_{ij}}_{\text{node}_i} & \mathbf{0} & \cdots & \mathbf{0} & \underbrace{\mathbf{B}_{ij}}_{\text{node}_j} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix}$$

Here $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ are the derivatives of the error function with respect to $\mathbf{x}_i$ and $\mathbf{x}_j$. This shows that even though there are T (corresponding to the number of nodes) elements in J_{ij}, only *two* are non-zero for each constraint. As a consequence, the structure of the block matrix $\mathbf{H}_{ij}$ in Equation 3.22 will be

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & \vdots & & \vdots & \iddots \\ & \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{A}_{ij} & \dots & \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{B}_{ij} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{A}_{ij} & \dots & \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{B}_{ij} & \\ \iddots & \vdots & & \vdots & \ddots \end{pmatrix}$$

In the above matrix, the zeros have been omitted for simplicity. The matrix $\mathbf{H}_{ij}$ corresponding to a constraint between nodes i and j has only *four* non-zero elements. Similarly the vector $\mathbf{b}_{ij}$ for each constraint will be sparse as shown:

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \\ \vdots \end{pmatrix}$$

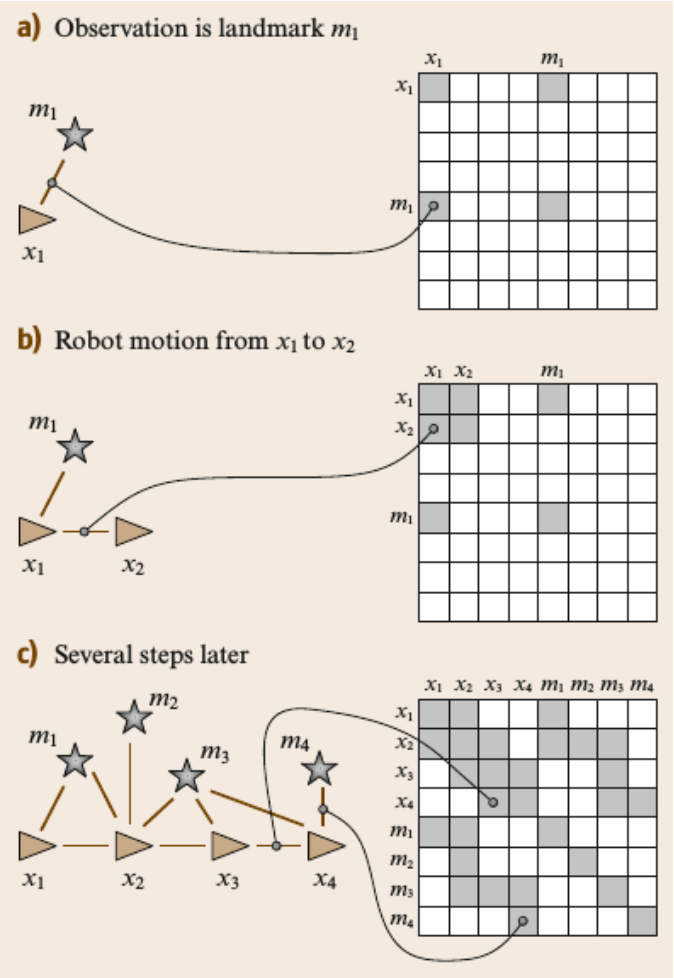


Figure 3.4: Illustration of the Graph Construction

The left diagram shows the graph construction. Robot pose and landmark observed are added as nodes to the graph. The right diagram shows how the constrains fill up the Information Matrix.

Source: [41]

The matrix $\mathbf{H}$ is an information matrix of the system, since it is obtained by projecting the measurement error in the space of the trajectories via the Jacobians (Equation 3.22) [1]. The matrix $\mathbf{H}$ and the vector $\mathbf{b}$ are obtained by summing up $\mathbf{H}_{ij}$ and $\mathbf{b}_{ij}$ of all constraints respectively (Equation 3.25). Each edge contributes to the blocks $\mathbf{H}_{[ii]}$, $\mathbf{H}_{[ij]}$, $\mathbf{H}_{[ii]}$, and $\mathbf{H}_{[jj]}$ of $\mathbf{H}$ and to the blocks $\mathbf{b}_{[i]}$ and $\mathbf{b}_{[j]}$ of the coefficient vector $\mathbf{b}$. If a block corresponding to two nodes is zero, there is no constraint between them. Now the information matrix $\mathbf{H}$ can be interpreted as the a matrix that defines the 'strength' of the links between nodes. Higher value of the block denotes a 'stronger' link. These links are more prominent, which means that their relative configurations have less tendency to change during graph optimisation. As mentioned, the blocks corresponding to unconnected node-pairs in $\mathbf{H}$ and $\mathbf{b}$ will be zero. Therefore their structures are sparse by construction. Figure 3.4 shows how each new link creates a new block in the information matrix $\mathbf{H}$.

An additional optimization possibility in the graph-based SLAM algorithm is to compute only the upper triangular part of $\mathbf{H}$, since it is symmetric. Most implementations of graph-based SLAM techniques in literature make use of these computational 'short-cuts' (sparsity and symmetry) to solve the optimisation problem [32, 33].

The Graph Optimisation Algorithm

Using the knowledge of the structures of the matrices involved, the steps involved in the graph optimisation of the graph-based SLAM problem is summarized in Algorithm 2.

As noted previously, the error of a constraint $\mathbf{e}_{ij}$ depends only on the relative position of the connected poses $\mathbf{x}_i$ and $\mathbf{x}_j$. Accordingly, the error $\boldsymbol{F}(\mathbf{x})$ of a particular configuration of the poses $\mathbf{x}$ is invariant under a rigid transformation of all the poses. This results in Equation 3.26 being under-determined. To numerically solve this system it is therefore common practice to constrain one of the increments $\Delta\mathbf{x}_k$ to be zero. This can be done by adding the identity matrix to the kth diagonal block $\mathbf{H}_{[kk]}$. Without loss of generality, the first node is kept fixed in Algorithm 2. An alternative way to fix a particular node of the pose-graph is to suppress the kth block row and the kth block column of the linear system in Equation 3.26 [1].

Algorithm 2 Graph Optimisation using Gauss-Newton Minimisation

Require: Initial Guess $\breve{\mathbf{x}}$; and constraints $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \Omega_{ij} \rangle\}$
Ensure: $\mathbf{x}^*$: new solution; $\mathbf{H}^*$: new information matrix

// iterate till no substantial improvements attained
while ¬converged **do**
  $\mathbf{b} \leftarrow 0 \quad \mathbf{H} \leftarrow 0$
  for all $\mathbf{e}_{ij} \in \mathcal{C}$ **do**
    // compute Jacobians of error function
    $\mathbf{A}_{ij} \leftarrow \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\Big|_{\mathbf{x}=\breve{\mathbf{x}}} \quad \mathbf{B}_{ij} \leftarrow \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\Big|_{\mathbf{x}=\breve{\mathbf{x}}}$
    // compute the contributions of the constraint to the linear system
    $\mathbf{H}_{[ii]} \mathrel{+}= \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{A}_{ij} \quad \mathbf{H}_{[ij]} \mathrel{+}= \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{B}_{ij}$
    $\mathbf{H}_{[ji]} \mathrel{+}= \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{A}_{ij} \quad \mathbf{H}_{[jj]} \mathrel{+}= \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{B}_{ij}$
    // compute the coefficient vector
    $\mathbf{b}_{[i]} \mathrel{+}= \mathbf{A}_{ij}^T \Omega_{ij} \mathbf{e}_{ij} \quad \mathbf{b}_{[j]} \mathrel{+}= \mathbf{B}_{ij}^T \Omega_{ij} \mathbf{e}_{ij}$
  end for
  // keep the first node fixed
  $\mathbf{H}_{[11]} \mathrel{+}= \mathbf{I}$
  // solve linear equation using sparse Cholesky factorization
  $\Delta\mathbf{x} \leftarrow \text{solve}(\mathbf{H}\Delta\mathbf{x} = -\mathbf{b})$
  // update parameters
  $\breve{\mathbf{x}} \mathrel{+}= \Delta\mathbf{x}$
end while
$\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$
$\mathbf{H}* \leftarrow \mathbf{H}$
// release fixed node
$\mathbf{H}_{[11]} \mathrel{-}= \mathbf{I}$
return $\langle \mathbf{x}^*, \mathbf{H}^* \rangle$

A Simple Octave Implementation

A Matlab/Octave implementation of the graph-based SLAM algorithm was developed to study and test the back-end working of graph-based SLAM using least squares. The code is developed from the basic framework and sample datasets provided in [38]. The algorithm

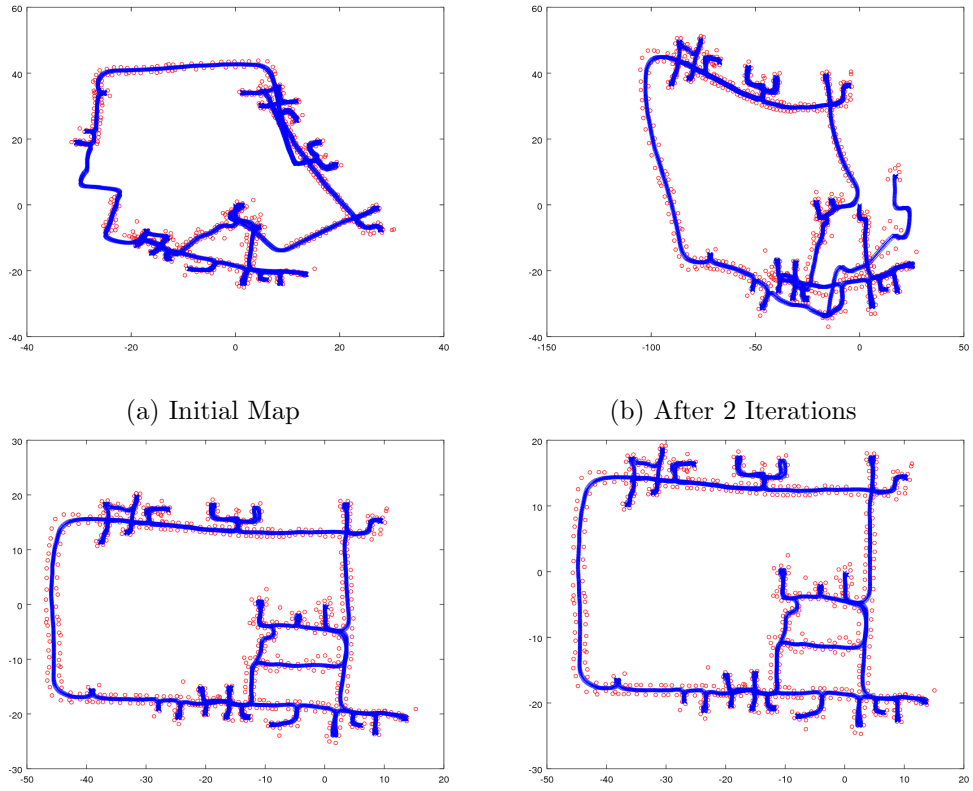
was developed to solve 2D SLAM problems when the odometry and measurements are provided.

The algorithm takes as input the details of the nodes and the constraints of the graph. The input contains the following data:

(a) *Nodes:* The nodes are the poses of the robot and the positions of the landmarks in the 2D world. The robot pose is defined by its 2D position in the world frame and an orientation with respect to one axis of the coordinate frame (x_p, y_p, θ_p), and the position of the point-landmark is defined by its 2D coordinates in the world (x_l, y_l).

(b) *Edges:* Each edge contains the details regarding the identities of the nodes it connect, the sensory measurement relating them, and the information matrix for the measurement. The measurement for a pose-landmark constraint is of the format $(x_{l(r)}, y_{l(r)})$, which describes the position of the landmark as observed from the robot's local frame. Similarly, the measurement of the pose-pose constraint will be the pose of the second robot pose in the frame of the first $(x_{p(r)}, y_{p(r)}, \theta_{p(r)})$. The constraints are defined using the relative transformation between the poses (or the pose and the landmark), and the information matrix relating them.

The algorithm works by minimising the error between the predicted measurement and the actual measurement, as explained previously. It follows the Gauss-Newton Error minimisation described above, with the original noisy odometry poses as the initial guess. The results of the algorithm at the end of certain iterations for the DLR dataset provided in [38], is shown in Figure 3.5.



(a) Initial Map

(b) After 2 Iterations

(c) After 5 Iterations

(d) At Convergence (10 Iterations)

Figure 3.5: Results of the Octave Implementation of Graph-Based SLAM at Various Stages of the Algorithm

The red circles indicate the estimated positions of the landmarks.
Dataset: DLR [38]

Chapter 4

SLAM Using Vision

Visual SLAM (or vSLAM) has been extensively studied in the past couple of decades. The main reasons for the surge in interest towards vision-based localisation and mapping techniques can be attributed to the rich visual information available from the relatively cheaper and light-weight video sensors, as compared to other sensors such as laser or sonar. However, the trade off is a higher computational cost and the requirement for more sophisticated algorithms for processing the images and extracting the necessary information. Fortunately, due to the advances in CPU and GPU technologies, the real time implementation of the required complex algorithms is not an insurmountable problem any more. In fact, a variety of solutions to the SLAM problem using different visual sensors including monocular [2, 42], stereo [43], omni-directional [44], 3D [45], and combined colour and depth (RGB-D) [46, 19] cameras have been proposed.

Visual SLAM is a particular case of a technique known as *Structure From Motion* (SFM) that tackles the problem of reconstruction of both the 3D structure of the environment and camera poses from sequentially ordered image sets [47, 48]. A typical vSLAM algorithm works by detecting and tracking well-localised features (with known 3D positions) across the image frames. The assumption of rigidity in the scene is then used to assert that the feature motion observed in the images is only due to the movement of the camera relative to the unknown (but static) 3D geometry or 'structure' of the features. This permits solutions for both the motion of the camera and the locations of the 3D features to be obtained [2]. Long chains of these frame-to-frame motion solutions can be stitched together to produce an estimate of a complete camera trajectory and full 3D map of all the features observed. The quality of the overall solution can then be refined by constraint-optimisation (graph-optimisation, also called '*bundle-adjustment*' in computer vision).

One of the earliest successful attempts at using vision for localisation is DROID in the late 1980s [49], where the algorithm estimates the ego-motion of the camera from the features matched across the images. The robotic community had turned away from vision-based SLAM techniques due to the more complicated algorithms and higher computational complexity, before the recent advances in the computing power of machines beckoned them back. MonoSLAM [50] is a more recent (2003, 2007) approach to the vSLAM problem, which created a sparse map using only 'high-quality' features from the scene. A real-time particle-filtering method for vision based tracking of a hand-held camera is described in [51]. Another breakthrough in vision-related tracking came with the development of the Parallel Tracking and Mapping (PTAM) framework [52], which split the processes of tracking and mapping into two separate computation threads. This method was shown to be more effective than previous tracking methods such as MonoSLAM in terms of speed and robustness.

Modern vSLAM algorithms are much more robust and reliable in terms of speed and

accuracy of mapping. ORB-SLAM is a popular and accurate solution to the vSLAM problem, which computes the camera trajectory and a sparse 3D reconstruction of the scene. Methods such as DTAM (Dense Tracking and Mapping) [53] creates dense representations (Section 4.2) of the environment by making use of all the pixels in an image, instead of selected features. LSD-SLAM (Large Scale Direct SLAM) [54] is an example of a fully direct (Section 4.2) vSLAM approach which can create large-scale, semi-dense maps. Predictive algorithms have also been implemented to improve the mapping process in SLAM. For instance, SLAM++ [55] brings object recognition to the front of SLAM, and directly builds a map at the higher level to benefit from strong predictions immediately.

A temporally scalable visual SLAM technique using a reduced pose-graph representation is described in [42]. The method improves the efficiency of computation by avoiding redundant frames and not using marginalisation to reduce the graph built. A graph-based approach for localising a 6-DoF robot using a 3D camera in a feature-sparse environment is demonstrated in [45]. The authors also prove that graph-based methods show more consistent performance than the EKF-based method in visual feature-rich and feature-sparse environments.

The remainder of this chapter will discuss the most important components in the front-end of a typical vSLAM system: visual odometry, mapping and loop-closure detection.

Visual Odometry

Visual odometry (VO) is the process of estimating the ego-motion (the 3D motion) of an agent using only the inputs of a single or multiple calibrated camera(s) attached to it [48, 47]. For a SLAM system that does not make use of any sensor other than a camera, visual odometry is the motion model. It is generally done by estimating the camera pose at each frame. The algorithm begins with known 2D-3D correspondence features, and tracks them across consecutive images. Computer vision provides the solution to obtain the pose of the camera from these correspondences at each frame, using projective geometry [56]. When the tracked features are lost (or are too few in number), new features are detected and their 3D positions in the world estimated by triangulation.

Visual odometry is considered superior to wheel odometry as it is not affected by wheel slippage, uneven terrains or other adverse conditions [47]. Additionally, in GPS-denied environments such as the interior of a building, or underground, VO has utmost importance. Although VO does not solve the drift problem, researchers have shown that VO methods perform significantly better than wheel odometry and dead reckoning techniques [57] while the cost of cameras is much lower compared to accurate IMUs and laser scanners. The price to pay, however, is not necessarily small. VO requires the environment to be textured, illuminated and static. The performance of the VO algorithm also depends heavily on the calibration of the camera, its motion, resolution, frame-rate etc.

This report does not include any further discussions regarding VO as it was a part of the second-term mini-project (Visual Odometry and PID Controller for an Aerial Robot). The mini-project report contains the detailed discussions regarding the development and implementation of the feature-based VO algorithm that has been used as the motion model for this project.

Mapping in Visual SLAM

In a visual SLAM system, the camera is the input sensor and the image captured is the information obtained. In this context, mapping would mean converting this image into a representation of the actual scene which can be used later. Depending on the type of visual sensor (camera) used, the mapping methods vary. When using a monocular RGB

camera, two separate types of classifications of vSLAM mapping are seen in literature: (i) sparse or dense, (ii) direct or indirect.

Sparse and Dense Methods

Based on the regions of the image used for mapping, visual mapping techniques are classified as sparse or dense. Sparse methods use only a small selected subset of the pixels in an image frame, while dense SLAM systems use most or all of the pixels in each received frame. Therefore the maps generated by the two methods are very different.

The maps generated from sparse methods are basically point-clouds, which are a coarse representation of the scene. PTAM [52] is a popular example of using sparse point representations of the scene for localisation. Most traditional methods used sparse representations for creating the map [49, 50, 51].

On the other hand, dense maps provide much more detail about the scene. The world is modelled as dense surfaces using whole-image alignment. However, the usage of more number of pixels than sparse methods demands more powerful hardware (most current dense SLAM systems require a GPU). DTAM [53] is a good example of a vSLAM implementation using dense mapping. Dense representations are commonly used when a depth camera is available [58, 46]. Semi-dense representations which do not use all the pixels, but only contrasting regions in the image, have also been successfully implemented [54, 59].

Direct and Indirect Methods

This classification is based on the way that SLAM systems utilise the information received from the images. Indirect methods first attempt to extract image features, and then make use of them to locate the camera and build the map. These features can be simple geometric features such as corners or edges, or more sophisticated feature descriptors such as SIFT, SURF, etc. The ORB (Oriented FAST and Rotated BRIEF) feature detector [60] was used for creating the indirect SLAM method called ORB-SLAM [61].

Direct methods, on the other hand, make use of pixel intensities directly, rather than extracting features. They try to recover the environment depth and structure and the camera pose through an optimisation on the map and camera parameters together [62]. LSD-SLAM [54] and DTAM [53] are examples of successful implementations of the direct SLAM approach.

As the feature extraction process can take a lot of time, direct methods potentially allow more time for other computations while maintaining the same frame rate as indirect methods. However, indirect feature-based methods provide better tolerance towards varying illumination as they do not use the pixel intensities directly. A very efficient and robust method called semi-direct method, which combines the advantages of both methods, has been demonstrated in [63].

Loop Closure Detection in vSLAM

The naive approaches adopted in early SLAM works to detect loop closures simply perform a nearest neighbour statistical gate on the likelihood of the current measurements given the map and pose estimates [64]. Basically, these techniques rely heavily on the pose estimate, and looks for loop closure in regions of the map that are within the uncertainty ellipsoid of the current pose estimate. However, if there is considerable error in the pose estimate - as is often the case when following a long trajectory - while the robot is in an already mapped area, the likelihood of measurements being explained by the pose and map estimate is very small [65]. Therefore, searching only in the neighbourhood of the

vehicle is not robust in the face of gross pose error. Even a small heading error over long linear traversals quickly leads to substantial position errors.

This is especially relevant for monocular visual SLAM systems which use the same sensor (monocular camera) for obtaining the pose estimate (via visual odometry), as well as the observations (image features) for building the map. It has been shown that due to liberalization and perception errors in monocular visual SLAM, the gross error in pose estimates can be so bad that the true location lies up to three times outside the bound of the pose uncertainty [65]. In such cases, it is clearly more effective to compare the current observation with all the past observations to detect loop closures, independent of the pose estimate.

Depending on where the data association for detecting the loop closure is done - in the metric map space or in the image space - the methods for detecting loop closure in monocular SLAM can be broadly divided into three categories:

(a) *Image-to-image Matching:* Correspondences are sought between the latest image from the camera and all/some of the previously seen images [27, 28, 29].

(b) *Image-to-map Matching:* Correspondences are sought between the latest frame from the camera and the features in the map [30].

(c) *Map-to-map Matching:* In this method, instead of comparing a single frame to another frame or the whole map, a sequence of image is used to form a local map, and correspondences are sought between the local map and the complete global map [31].

In [66], the authors compare methods from the three categories and conclude that the map-to-map method gives the best performance since it can prune out more false positive loop closure detections. In this project, however, the image-to-map method was not chosen due to its higher complexity. It requires separate randomised ferns classifier for each submap and has to cycle through submaps when attempting loop closure [30]. Instead an image-to-image detection method was adopted, and its ability to prune false-positives was improved by adding a few extra checks (Section 5.2.1). The improved implementation of the image-to-image matching used for this project is much simpler to implement and guarantees zero false detections.

The 'Bag-of-Words' Technique

An obvious method that comes to mind for finding loop closures using images is feature matching. Extracting prominent features in an image and matching them to the features in another image is a fail-safe way of making sure the two images contain the same object(s). However, loop closure detection involves searching through the entire history of image frames. Each new frame obtained has to be compared with every previous frame in the mapping history. As the number of frames grows, this naive solution quickly becomes computationally intractable. This issue is tackled by creating a *visual vocabulary* before the mapping and comparing images using the *visual words* in them. In literature, such methods are called 'bag-of-words' methods.

The bag-of-words image matching technique was developed by Sivic & Zisserman in 2003 [67] with the intention of retrieving "those key frames and shots of a video containing a particular object with the ease, speed and accuracy with which Google retrieves text documents (web pages) containing particular words". It works by first extracting features from a set of few images that are similar to the actual scene, and vector-quantizing their descriptors into clusters to form *visual words*. This way a *visual vocabulary* is created. Then when a new frame is observed each descriptor of that frame is assigned to the

nearest *word*, and this immediately generates matches for all frames in the map history. Each image is represented by a vector of *word frequencies*, which is just a vector of weighted counts of each *word* from the *vocabulary* in that image (Figure 4.1).

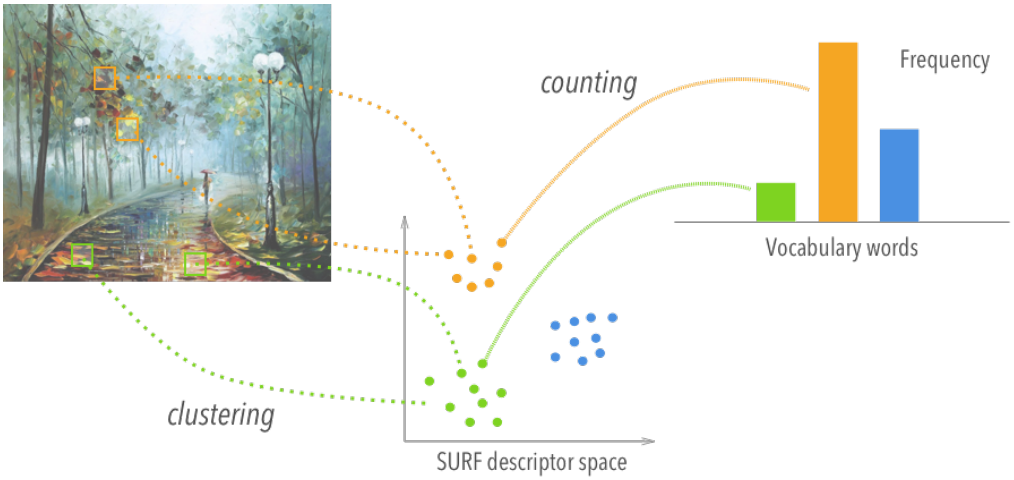


Figure 4.1: Illustration of the Bag-of-Words Image Representation

Each feature in the image is clustered to the nearest 'word' in the 'vocabulary'. The number of times each 'word' occurs is counted. The image is representated by the counts of 'words' from the 'vocabulary'.

Retrieved from: `http://nicolovaligi.com/bag-of-words-loop-closure-visual-slam.html`

Several modifications of the original bag-of-words method are available in literature now. Quicker feature extraction and description techniques improved the efficiency of the image search [68]. Binary feature detectors and descriptors further improved the speed of computation, giving almost real-time loop closure detection [69]. The bag-of-words technique was approached in a probabilistic format using Bayesian filtering in [70]. In [71], the authors modified the method to build a vocabulary tree that discretises a binary descriptor space, which is then used to speed up the correspondences for geometrical verification.

The benchmark in loop-closure techniques for vSLAM is probably the Fast Appearance-Based Mapping (FAB-MAP) algorithm [5]. It is a probabilistic approach to the bag-of-words technique. It represents images using a bag of words, and utilises a Chow Liu tree to learn offline the words' co-visibility probability [71]. During the learning process, the algorithm is trained with a few images which are similar to the images in the actual image dataset, to create a *visual vocabulary* as described above. Later, during mapping, if a new image is found to come from a previously visited scene, it gives the most probable image match out of all previous images. If the image is not a loop closure, it identifies it as a new place in the map. The authors claim this algorithm to be "effectively a SLAM system in the space of appearance". The probabilistic approach allows to explicitly account for perceptual aliasing in the environment. Therefore identical but indistinct observations receive a low probability of having come from the same place. This is achieved by learning a generative model of place appearance.

The framework was further developed (FAB-MAP 2.0) to provide efficient large-scale implementation of loop closure detection over a 1000km trajectory [6]. The modifications include the use of an indirect inverted index retrieval architecture, and switching to sample-based representation of locations, rather than appearance-based. These modifications allow a fully sparse evaluation, leading to much faster computation.

An open-source implementation of FAB-MAP is available in OpenCV, and a modification of this library was used for loop closure detection in this project (Section 5.2.1).

Chapter 5

Implementation

Setting Up

Hardware

The robot platform on which the graph-based vSLAM algorithm was implemented is the AR.Drone 2.0. It is a relatively cheap quadcopter with a wide range of on-board sensors. It weighs less than 500 grams and can be controlled wirelessly using various mobile and desktop interfaces. It has a motherboard containing the main processing unit (an ARM Cortex-A8 running at 1 GHz), two cameras, and a Wi-Fi module[72]. One of the cameras is oriented vertically, pointing to the ground. It has a frame rate of 60 frames per second (fps) and a resolution of 320 × 240 pixels, and is used for estimating the horizontal speed of the drone. The second camera is pointing forwards and can capture images of 720p resolution at 30 fps. A second board, the navigation board, contains all the sensors that can be used for the state estimation of the quadrotor. Although the sensors include GPS, an IMU consisting of a gyroscope and accelerometer, pressure sensors, ultrasound sensors, and a barometric sensor, only the cameras were used for the project; the intention being to solve the SLAM problem using just vision.

The actual SLAM algorithm was implemented and run offline on an HP laptop with an i7-5500U CPU @ 2.40GHz, 6GB RAM running Ubuntu 14.04. It was also used for remotely controlling the AR.Drone and for running the PID controller for the drone required by it for trajectory following.

The motion of the quadrotor in action was also captured using a Qualisys Motion Capture System. The motion capture system consists of 12 Oqus cameras placed around the workspace, tracking speacially-designed markers placed on the drone, at the rate of upto 10,000 fps [73]. The trajectory tracked by the motion capture system is used as the ground truth for evaluating the trajectory estimated by the developed vSLAM algorithm. The motion capture system was also used to define a coordinate system in the workspace, which is used as the world frame for the project. An image of the workspace is shown in Figure 5.1.

The Scene Setup and Modification to the VO Algorithm

As mentioned in Section 4.1, the scenes to be mapped have to be highly textured for the proper visual odometry[1]. Hence, the surfaces to be mapped were covered with pages from

[1]It was shown in the mini-project report that the performance of the VO algorithm relies on many parameters which have to be specifically tuned for each dataset. Accordingly, they were tuned for each dataset used in this project to obtain the best possible result for VO. However, the tuning procedure or the optimal values of these parameters are not included in this report as they come under the scope of the mini-project.

newspapers for creating a textured scene.

To begin pose-estimation, the VO algorithm requires beforehand the 3D world positions of a few points in the first image frame. The synthetic datasets used for the mini-project (ISMAR Offline Tracking Competition, 2015) provided these correspondences in the form of a readable file. So the algorithm could use this information to begin trajectory estimation. Obviously, such correspondence information will have to be manually created while building a new original dataset. This meant that the positions of some features in the environment has to be known. More importantly, the algorithm has to detect and identify these (and only these) features in the first image.

Using a checkerboard whose dimensions are known is the simplest way to tackle both the problems. A function for detecting checkerboard corners in an image is available freely from OpenCV [74]. The 3D positions of all corner points on a checkerboard can be easily obtained using basic geometry if the number of squares per column and row, and the 3D positions of at least three of its four outer corners are known. By combining these two methods, a new way of initialising the VO algorithm was developed - by placing a checkerboard in the beginning of the scene (Figure 5.1). Having obtained the positions of three corners of the checkerboard (using the motion capture system and markers), the drone can then be made to traverse in the workspace, capturing the textured environment as it moves.

Figure 5.1: Workspace

The scene to be mapped and the drone used for capturing the scene can be seen. The checkerboard used to initialise visual odometry is placed at the beginning of the scene. Two of the tracking cameras used by the motion capture system are also visible at the top of the image.

Defining Input and Output for the vSLAM Algorithm

The performances of the developed vSLAM algorithm on three different datasets have been evaluated in this report. Two of them were created by remotely controlling the drone to move around in the environment, while for one the drone was made to move autonomously along a predefined trajectory using the PID controller (developed as part of the second-term mini-project: Visual Odometry and PID Controller for an Aerial Robot). Each dataset consists of the sequence of images captured by the moving drone, the trajectory given by the motion capture system (ground truth), the calibration of the camera, and the positions of three checkerboard corners.

The sequence of images, the camera calibration and the checkerboard corner positions form the input for the developed vSLAM algorithm. The output is the final estimation of the trajectory and the corresponding map of the environment. The accuracy of the trajectory (and thereby, the performance of the vSLAM system) can then be evaluated by comparing it with the ground truth in the corresponding dataset.

The Front-End

Loop Closure Detection

The FAB-MAP algorithm has become the standard when it comes to loop closure detection. However, its robustness decreases when the images depict very similar structures for a long time, which can be the case when using frontal cameras [37]. It detects false

loop closures when the types of objects in the sequence are similar. Adding a false loop closure constraint can have serious consequece in the final graph. Graph optimisation based on the wrong constraints would give very wrong trajectory and a distorted map. Moreover, FAB-MAP requires a sparse image sequence (considerable camera translation between subsequent frames), while a dense sequence is required for the proper working of the visual odometry algorithm.

Therefore, several modifications were made to the open-source FAB-MAP algorithm to detect loop closures in this project. This was done to ensure no false loop closures were detected at all. The modifications tackle FAB-MAP's problem of detecting false loop closures when the objects in the scene are similar, and also allows the algorithm to utilise only a sparse subset from the dense image sequence used by the visual odometry.

Every time a new node is added to the graph, the loop closure detection algorithm checks for matching images in its memory. The modifications made to the FAB-MAP algorithm are:

(i) *Sparse Test-Data*: Each time an image is observed the FAB-MAP algorithm adds it to its 'Test-Data', which contains all the previous images to which a new image has to be compared. The images are stored in their bag-of-words representations described in Section 4.3. When a new image is observed, the algorithm gives a probability value for each image in its Test-Data. If the value for an image in the Test-Data is high enough, it is considered to be a loop closure for the new image. If none of the images have a significant probability value, the algorithm identifies it as an image of an unexplored region in the map.

Since the image datasets for the developed vSLAM algorithm should be very dense for it to run smoothly, consecutive images in the sequence have very little motion between them, and have identical objects/regions in the images. If the original FAB-MAP algorithm was allowed to run on such a dataset, this would result in many images having the same probability of being a match to a new image. Moreover, if there are too many matches, the FAB-MAP algorithm will try to divide the probability equally, and since the total of induvidual probabilities should sum up to one, this would mean that the values would be too small to be significant, and would be indistinguishable from the wrong matches.

To overcome this issue, the first modification made to the FAB-MAP algorithm ensures that it adds only every m^{th} image to its test-data. The value of m depends on the density of the sequence (number of images after which there is considerable difference in the objects/regions in the image), frame-rate etc. This way, the Test-Data is sparse, and contains dissimilar images which are more distinguishable from each other. In addition to making the loop closure matching easier, this step also reduces the computation time required due to the highly reduced number of images to compare with.

(ii) *All-but-one Test-Data*: Even after ensuring that only every m^{th} image makes up the FAB-MAP Test-Data, the high density of the image sequence can still cause issues in detecting loop closures. Consider a situation where an image was just added to the FAB-MAP Test-Data. The images that are processed right after this instant will be good matches to this latest addition to the Test-Data, and will identify it as a loop closure. To avoid this problem, the second modification made to the FAB-MAP algorithm ignores the last image in the Test-Data while looking for a loop closure match.

(iii) *Repeated-Matches Verification*: As mentioned previously, the original FAB-MAP algorithm detects too many false loop closures when the objects in the image sequence

are similar. Since the datasets for this project were created out of an environment littered with newspapers, there is little variation in the *type* of objects in the scene. Due to this, over 85% of the detections made by the original FAB-MAP algorithm on this dataset were false positives. The third modification made to the loop closure detection algorithm considerably reduces these false detections.

This modification takes advantage of the denseness of the image sequence by checking for repeated matches. If subsequent images are matched to the same image in the Test-Data, then the loop closure is more likely to be true. Therefore this modification considers a detection to be true only if at least r consecutive images are matched to the same image in the Test-Data. This modification is feasible as subsequent images have very little variation due to the image density, and an image in the Test-Data will be a good match to a subset of consecutive images if there is a loop closure.

(iv) *Optical Flow Check*: When there is a relative motion between the scene and the camera, there is an apparent motion of the objects in the image. This pattern of apparent motion that is observed on the image due to the relative motion between the camera and the scene is called optical flow. Optical flow methods try to calculate the motion between the image frames [75]. The basic methodology of doing this is by assuming that all pixels in the image belonging to an object must move in the same direction across the images. The Lucas-Kanade (LK) algorithm is a popular optical flow calculation method. The method has been implemented previously for tracking features across subsequent images in the visual odometry algorithm for the mini-project[2].

The optical flow check is the final check for the loop-closure detection algorithm. If there is a good optical flow between the images, it means most of the features in one image are present in the other, and they have all moved in the same way (i.e., obtained from the same camera motion). This is a fail-safe way of checking for loop-closures. However, it has to be noted that checking for optical flow may fail if there is large rotation about the camera axis (more than about 90 degrees) between the images of the same scene. But owing to the fact that such a rotation about the front camera axis for a moving robot is generally unlikely, this is not a considered to be an issue.

The modifications were experimentally tested and proved to perform better than the original FAB-MAP algorithm in terms of filtering out false positives (Section 6.2). The steps involved in the final loop closure detection algorithm are formally shown in Algorithm 3.

Another way of verifying loop closures was implemented initially. It involved using the features in the current image, whose 3D positions in the world are known (from the VO algorithm), and checking for matches in the loop closure image after the 'Repeated Matches Verification' step. If it was a true loop closure, most of the features would have matches, and their new 3D positions can be estimated for that image. Then, a point-to-point model registration based on RANSAC can be used to estimate the best alignment for the matching 3D points. This would give the measurement required for creating a new constraint.

However, in practice, comparing feature points is computationally expensive as well as unreliable. Due to the similar appearance of the objects in the scene (texts and images on the newspaper pages), the method identified wrong features as matches (Figure 5.2a). The number of matches detected were also very less. This is the result of the correspondence

[2]Detailed description regarding the mathematics and implementation of the optical flow calculation using the LK method is not included in this report, as it has been discussed in the mini-project report.

Algorithm 3 Loop Closure Detection Algorithm for the Project

// Given the current image and the best match from FAB-MAP, the algorithm checks if it is a true loop closure each time a new node is added to the graph. This algorithm runs as a part of the vSLAM algorithm.

Require: New Image I_n (n: Node ID); Matched Image from Test-Data I_k (k: FAB-MAP Test-Data ID)

Ensure: y (Loop Closure Status (True/False))

// Setting the loop closure status to 'False'. It is set to 'True' only if all the checks are passed.

$y \leftarrow$ False

// Every m^{th} image is added to the Test-Data. (Also avoids using the latest member in Test Data for loop matching.)

if $n\%m == 0$ **then**
    add_to_FABMAP_TestData(b)
    b $\leftarrow I_n$
end if

// Resetting match_count if consecutive images are not matched to the same image in Test Data (if no matches are found in the Test Data, $k = -1$).

if $k == -1$ OR prev_loop_id $\neq k$ **then**
    match_count = 0
else
    match_count += 1

// If a match is repeated a minimum of r times, the images are checked for optical flow.

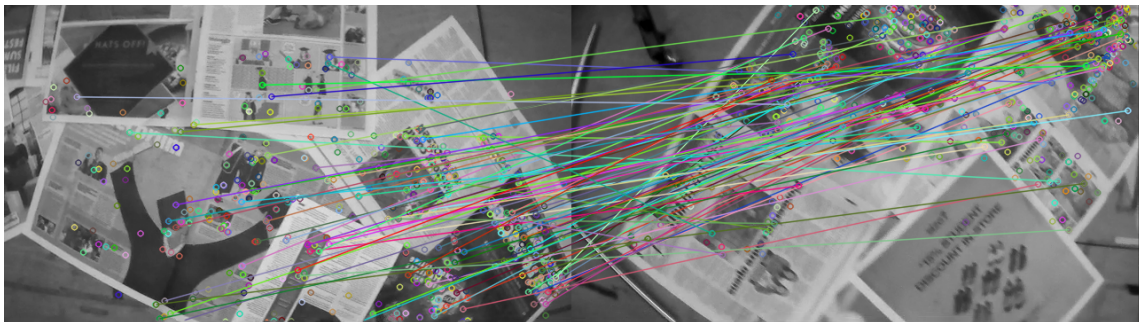
    if match_count $\geq r$ **then**
        flow_status = check_optical_flow(I_n, I_m)

// It is a true loop closure if there is optical flow between the images.

        if flow_status == True **then**
            $y \leftarrow$ True
        end if
    end if
end if
if y == True **then**
    add_loop_closure_constraint($n \rightarrow k$)
end if

problem (data association) in computer vision [76, 77]. The difficulty in data association between two image frames arise due to the possible change in scene illumination, change in view-point, detecting noisy points as features, obstruction of the features in the second image, presence of features that appear as edges or corners in one image but not in the second, etc. Using this wrong information for 3D alignment of the points would obviously give hugely erroneous measurement for the constraint. This would not only give the wrong estimate at that time frame, but would ruin the entire map after graph optimisation.

Moreover, feature-matching techniques search the entire image for each feature, whereas optical-flow methods search for features in the region where they are most likely to be (once a few good features have been detected) (Figure 5.2b). This reduces the computation time required considerably. Optical flow can also check if the matched features can be explained by the same relative camera motion between the images. Therefore, feature-matching was replaced with optical flow check in the final loop closure detection algorithm for this project.



(a) Feature Matching Method

(b) Optical Flow Method

Figure 5.2: Results of the Two Loop Closure Detection Methods

It can be seen that the feature matching technique finds too many false matches, and detects a false loop closure, whereas the optical flow method identifies very good loop closures (albeit a few false matches).

The modifications implemented for this project (Algorithm 3) are extensive checks to ensure that no loop closures are detected during the run of the vSLAM algorithm. This is because detecting even one false loop closure could irrecoverably distort the graph after optimisation (Figure 6.8). These modifications have sometimes failed to detect a loop closure (Section 6.2) due to its intense filtering. However, detecting another true loop closure after this failure can make up for it, and still optimise the graph reasonably well. Moreover, since the vSLAM algorithm depends on a single sensor, the measurements and the graph cannot be verified or recovered using other sensors. All these factors justify the decision to ensure zero false positives in the loop closure detection process, even if that meant failing to detect a few true positives.

The Graph - Nodes and Edges

Nodes

The nodes of a graph in a graph-based SLAM algorithm are usually made up of the robot poses and/or the landmarks in the map. However, when using the same sensor (camera) for obtaining the poses (motion model) and the map (measurement model), creating nodes out of both the poses and the landmarks is pointless, since one is estimated by assuming that the other is fixed (visual odometry assumes the features tracked in the images have fixed positions in the world). This is equivalent to having a fixed edge linking a robot pose to the landmarks observed by it at that point, which makes the edge a 'hard' constraint (zero covariance). Such hard constraints between nodes are not affected when the graph is optimised, and is hence unnecessary to keep such nodes in the graph. The map can be easily rendered in the end, by placing the landmarks relative to the corresponding

estimated poses in which they were observed.

Therefore, a node for the graph in this SLAM project is made out of the robot's poses. Each node contains information consisting of a unique ID for the node, the pose of the robot at that time-frame as given by the visual odometry, the image frame captured by the camera at that pose (for loop closure matching), and the 3D positions of the landmarks observed in that frame (for map generation).

Edges

Edges relate the nodes using their relative transformation in space as obtained from the measurements. Edges are created in two scenarios: (i) When the robot moves from one pose to another (odometry constraint - relates two consecutive nodes), and (ii) when the robot observes a loop closure (loop closure constraint - relates two nodes that are distant in time).

An edge (z_{ij}, Ω_{ij}) is defined by the measurement relating the two nodes i and j, and the corresponding information matrix defining its 'strength'. As mentioned previously, there is no closed-form solution for assigning an information matrix Ω_{ij} to an edge, and is done empirically for a particular algorithm or dataset [16, 37, 4]. The method of determining z_{ij} and the metric for computing Ω_{ij} for the constrains in this project is described below.

(a) *Odometry Constraint*

The measurement z_{ij} of an edge connecting two poses X_i and X_j should be the relative transformation obtained by aligning the observations acquired at the two poses. Since visual odometry achieves pose estimations at each time frame by doing exactly this, the relative transformation between the poses, both in terms of measurements and in terms of the odometry poses (predicted measurement), will be exactly the same ($z_{ij} = \hat{z}_{ij} = X_i^{-1} X_j$, where X_i and X_j are poses in the form of transformation matrices).

Visual odometry is known to drift inevitably over time. This can be considered as a continuous increase in the uncertainty of the pose estimate. Since Ω_{ij} is the inverse of the covariance matrix, the metric chosen for computing the information matrix for odometry constraints is

$$\Omega_{ij} = (c_t \times \mathbf{I})^{-1}$$

where c_t is a number that increases proportionally with time (or with the number of nodes in the graph), and $\mathbf{I}$ is the identity matrix. This metric gives the required reduction in the 'strength' of the odometry constraints over time. It depends only on time and is independent of the poses or image-matches between the nodes.

(b) *Loop Closure Constraint*

Once a loop closure is detected between a new node p and an old node q in the graph, the measurement z_{pq} between them is calculated using just the images (observations) captured at the two nodes, by performing the steps from the VO algorithm on them.

Knowing the 2D (pixel) and 3D (world) positions of a few features in the image of node p (from VO and mapping), these features are tracked from the image of p to that of q. This gives a new projection matrix for the image frame of node q, from which a the new camera pose can be estimated[3]. It has to be noted that this relative

[3]The steps involved are not described in detail as they are exactly the same as the VO algorithm. In fact, the relative measurement transformation for a loop closure in this project was obtained simply by giving the two images to the previously developed VO code. It returns a new pose estimate for the second image, from which the relative transform can be calculated.

transformation will generally not be the same as the relative transformation obtained by aligning the poses associated with the nodes, i.e., $z_{pq} \neq X_p^{-1} X_q$ (where X_p and X_q are the original poses associated with each node), due the drift in odometry and noise in the system. This error between z_{pq} and $\hat{z}_{pq} = X_p^{-1} X_q$ is what the back-end of the SLAM algorithm will try to minimise during graph optimisation.

$$e_{pq} = z_{pq}^{-1} \hat{z}_{pq} = z_{pq}^{-1} \left(X_p^{-1} X_q \right)$$

It can be seen that this is equivalent to the error term in Equation 3.18.

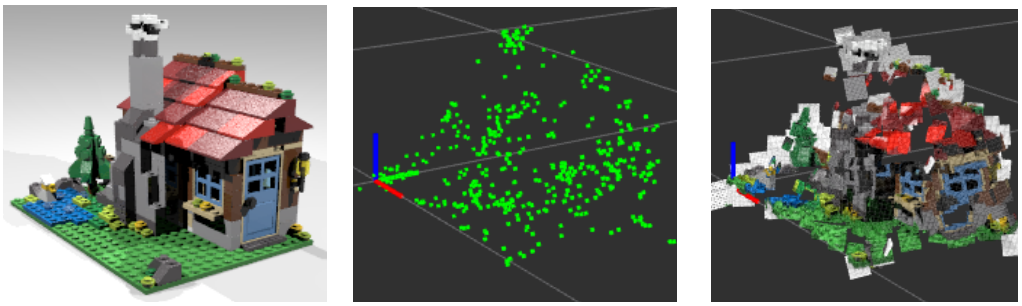
The optical flow check described in Section 5.2.1 is used as a means for obtaining the covariance for the loop closure constraint. The optical flow returns an error value for each of the features tracked, which denote how much they differed from their estimated positions. The variance of these values – given by $\sigma_{pq} = \frac{\sum_{f=1}^{P} (e'_f - \bar{e}')^2}{n}$, where e'_f is the error of the feature point f and $\bar{e}'$ is the mean error of the population – is used for determining the constraint's information matrix.

$$\Omega_{pq} = \mathbf{I} \frac{1}{\sigma_{pq}^2}$$

Mapping

An indirect, sparse mapping method was adopted for this project. The measurements at each node, i.e., the 3D positions of the tracked image features, were used to generate the map for this project. Each point was placed as a 3D point in the location estimated by the VO algorithm. If the pose of a node changes during graph optimisation, the map points corresponding to that node are also changed so as to maintain the relative position between the point and the robot pose. Hence, mapping is done by making use of 'hard' constraints between robot pose and landmarks, by assuming that there is no variance in the landmark positions estimated with respect to the robot pose.

Two slight variations of the sparse method were used for visualising the map. The first method involves placing the tracked landmarks as points in the world frame (Figure 5.3b). This is comparatively a more efficient and faster method, as compared to the second method.



(a) Original Image (b) Mapping Method 1 (c) Mapping Method 2

Figure 5.3: Comparison of the Two Mapping Methods

Dataset: International Symposium on Mixed and Augmented Reality (ISMAR) (2015) Offline Tracking Competition.

The second method makes use of the pixel information of a window around each tracked feature in an image, and maps the coloured patches into the world frame, giving a partial reconstruction of the scene (Figure 5.3c). Since the 3D position of the feature in the centre

of the window is known, all the other features in its window can be placed around it in the world frame. However, since the distance between points in a window for a feature that is physically closer to the camera will be less than that for a window around a feature that is actually far away. Hence the spread of the window in the world frame is proportional to the distance of the landmark from the camera. This method assumes that the points in the small window around the feature are actually close to each other in the world frame, and are not from objects far apart. This is clearly not a safe assumption since most tracked points are edges or corners. Additionally, the method requires storing more number of points and additional computation for positioning a window in the space accurately. So, this method is much slower than the first method, especially as the graph grows bigger.

The mapping methods are not evaluated or compared in this report as they were used only for visualisation, and are not directly related to the objectives of the project.

The Back-End

The popular g^{2}o (General Graph Optimisation) framework [33] performs the same graph optimisation procedure as explained in Section 3.3, but on smooth manifolds (Appendix A.3). It solves the minimisation problem efficiently by making use of the sparsity of the structures in the problem (Section 3.3.2). It provides solutions to several variants of SLAM and Bundle Adjustment problem.

Since g^{2}o has been widely used and accepted as an efficient graph optimisation framework [78, 54], reimplementing a similar framework would be pointless. It is available as an open-source C++ framework for optimizing graph-based non-linear error functions. Hence, the g^{2}o framework was chosen to be the back-end of the developed vSLAM algorithm for performing constraint optimisation.

Chapter 6

Algorithm Testing - Experiments and Inference

Dataset Description

Three different image sequences were used for testing the performance of the developed vSLAM algorithm. All were created by scattering newspapers over the area to be mapped for better texture. Due to the limited workspace area (region that can be tracked by the motion capture system for ground truth), and the inability of the VO algorithm to perform consistently over long distances, the sequences do not span over a few metres.

- *Dataset 1*: This is a sequence of the drone moving in a horizontal square trajectory, while capturing the images on the floor. The sequence starts and ends at the same point in space. The total distance of the trajectory is approximately 6 metres.
- *Dataset 2*: The drone is made to move around a horizontal circular trajectory repeatedly, while capturing the images on the floor. Since the sequence consists of the drone moving in a loop (three times), this dataset was used to test the performance of the loop closure detection. (Approximate circumference of trajectory: 8m)
- *Dataset 3*: The PID controller developed for the mini-project was used to make the drone move autonomously along a vertical square trajectory, while capturing images in front of it. The trajectory is approximately 6m long.

The working of the algorithm on these datasets are also included in the demo video that was created (Appendix B).

Accuracy of the Estimated Trajectory

Without Loop Closure Detection

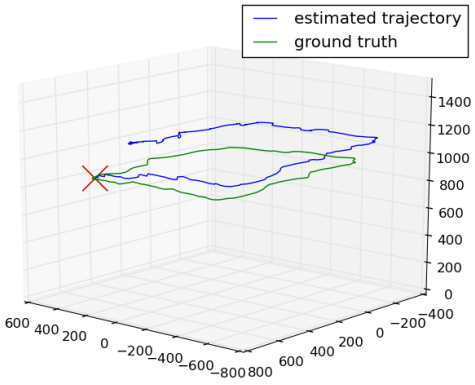
Errors in the trajectory estimation for the three datasets are discussed in this section. Accuracy is evaluated by measuring the deviation of the estimated trajectory from the ground truth along the three axes of the world coordinate frame. As mapping is done with respect to the poses of the robot in the trajectory (Sections 5.2.2 & 5.2.3), evaluating the mapping procedure separately is meaningless. The map will only be as accurate as the trajectory itself. The results of the algorithm for the three datasets and the inferences made from them are discussed below.

The estimated trajectory alongside the ground truth for the first dataset is shown in Figure 6.1a. The error in the trajectory can be attributed to the drift due to visual

odometry. It can be seen from Figure 6.1b that the drift is much more prominent along the z axis, which also aligns with the principal axis of the camera. This agrees with the knowledge that estimating depth using vision is more difficult than estimation of lateral distances [79, 48]. Depth estimation from 2D images is, in fact, one of the most prominent research areas in computer vision [79, 80, 63]. The total average error in trajectory (Euclidean) for the dataset was found to be 122.3 mm.

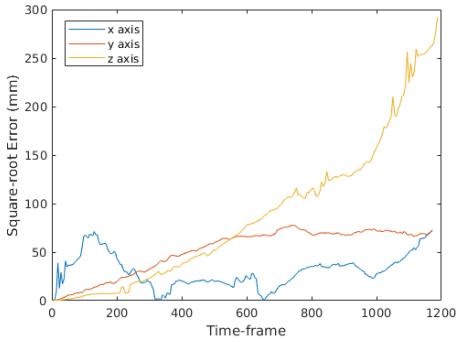
It can be seen from Figure 6.2 that the trajectory drift is similar in the case of Dataset 2 as well, when there is no loop closures detected. However, the drift is now more prominent, as the trajectory is longer than that of the previous dataset. When using only the first round around the circle without loop closures, the total average Euclidean error for the second dataset was obtained as 251.9 mm.

The algorithm failed to completely estimate the trajectory for Dataset 3. The visual odometry failed after some frames, resulting in the failure of the vSLAM algorithm as well. This was because of the unsteady motion of the drone as it traversed autonomously using the PID controller. The performance of the PID controller is not good enough to provide the required motion stability for the proper working of the VO algorithm. It can be seen in Figure 6.3a that the trajectory estimation failed before the sequence could be completed, resulting in an incomplete map (Figures 6.3c and 6.3d). The drifts in the estimate is also bigger and more erratic (Figure 6.3b).

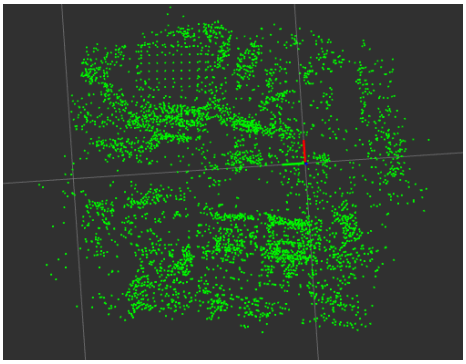


(Red 'X' marks the starting point)

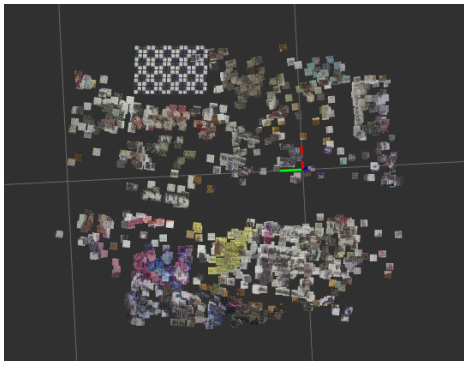
(a) Comparison of Estimated Trajectory and Ground Truth



(b) Trajectory Estimation Errors Along X,Y,Z Axes

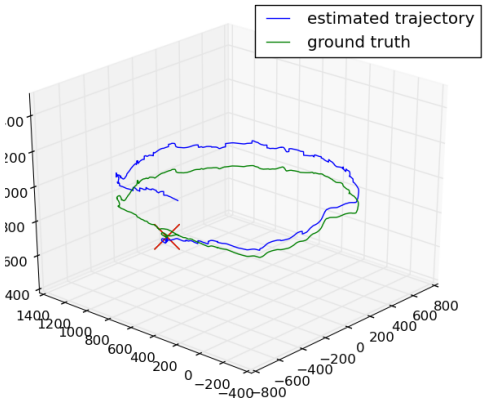


(c) Estimated Map (Mapping Method 1)



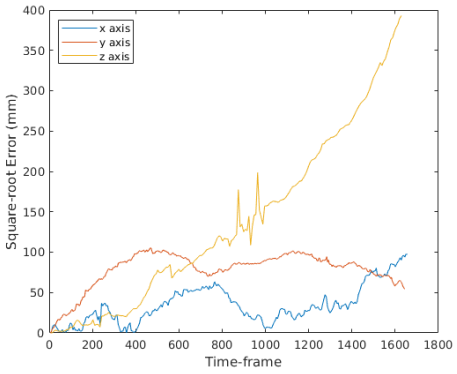
(d) Estimated Map (Mapping Method 2)

Figure 6.1: Dataset 1: Trajectory and Map Estimations

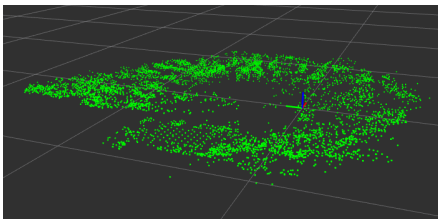


(Red 'X' marks the starting point)

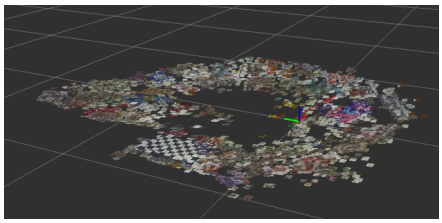
(a) Comparison of Estimated Trajectory and Ground Truth



(b) Trajectory Estimation Errors Along X,Y,Z Axes

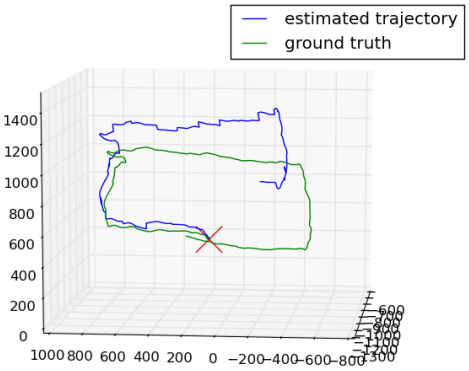


(c) Estimated Map (Mapping Method 1)



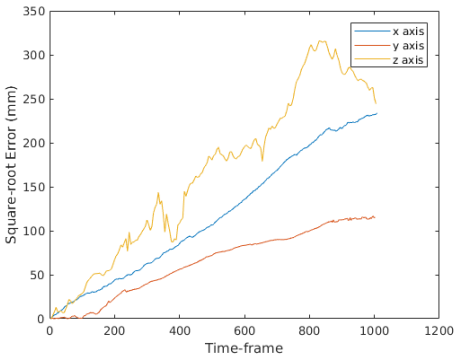
(d) Estimated Map (Mapping Method 2)

Figure 6.2: Dataset 2 Without Loop Closure: Trajectory and Map Estimations

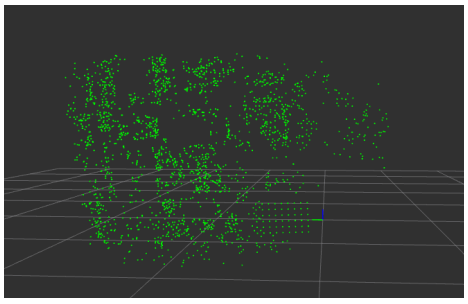


(Red 'X' marks the starting point)

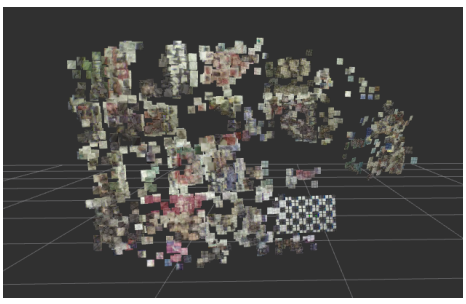
(a) Comparison of Estimated Trajectory and Ground Truth



(b) Trajectory Estimation Errors Along X,Y,Z Axes



(c) Estimated Map (Mapping Method 1)



(d) Estimated Map (Mapping Method 2)

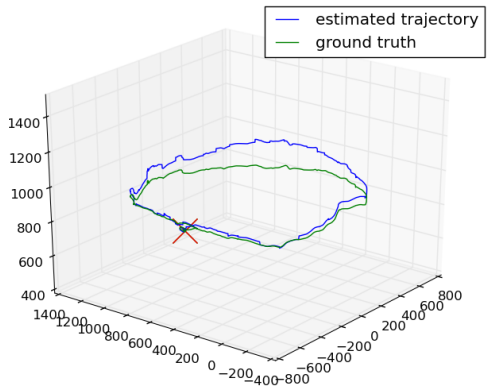
Figure 6.3: Dataset 3: Trajectory and Map Estimations

With Loop Closure Detection and Graph Optimisation

Partial Trajectory - One Loop Closure

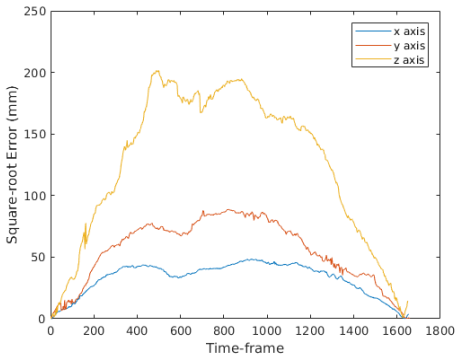
Dataset 2 was again tested to find the trajectory errors after loop closure optimisation of the graph. For this, the loop closure detection was turned on for the vSLAM algorithm and tested for the same portion of the trajectory as before (Figure 6.2).

The result is shown in Figure 6.4. It can be seen that the algorithm detected the loop closure and corrected the final pose (Figure 6.4a). Moreover, it has tried to fit the previous poses to better explain the observations made (Figures 6.4c & 6.4d). The drifts in the most recent poses along all the axes reduced considerably (Figure 6.4b), bringing down the average total trajectory error to 62.7 mm (from 251.9 mm, reducing the error by **75.11%**).

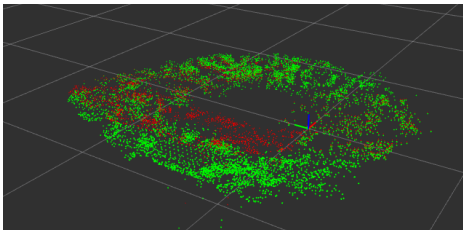


(Red 'X' marks the starting point)

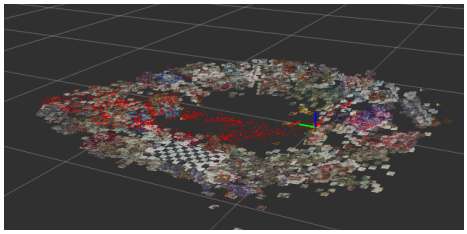
(a) Comparison of Estimated Trajectory and Ground Truth



(b) Trajectory Estimation Errors Along X,Y,Z Axes



(c) Estimated Map (Mapping Method 1)



(d) Estimated Map (Mapping Method 2)

The red points in the map indicate the previous map estimates (map before optimisation).

Figure 6.4: Dataset 2 With Loop Closure: Trajectory and Map Estimations

Full Trajectory - 16 Loop Closures

The algorithm was then tested on the full dataset (3 loops around the circular trajectory). Graph optimisation is triggered each time a new loop closure is detected. Figure 6.5 shows the total average error in the full trajectory estimated at each time-frame, for a portion of the dataset. It can be seen that the there is a dip in the total average error each time a loop closure is observed. The figure shows the detection

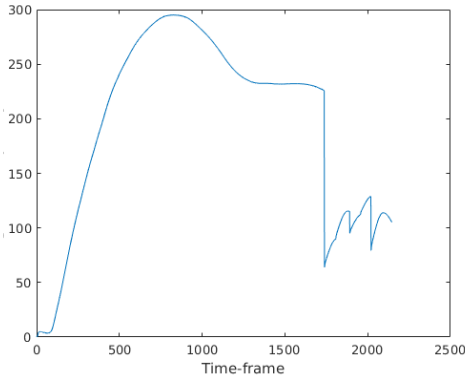


Figure 6.5: Average Total Error in the Estimated Trajectory at Each Time-Frame

of the first three loop closures (indicated by the vertical regions in the graph). The first loop closure is observed when the drone reaches near its starting point. Loop closures are then observed more frequently as the path has already been traversed by it before.

The error in the pose at each time frame (Euclidean distance from the ground truth) for the entire dataset is shown in Figure 6.6. It compares the difference in the errors for the algorithm with and without loop closure detection[1].

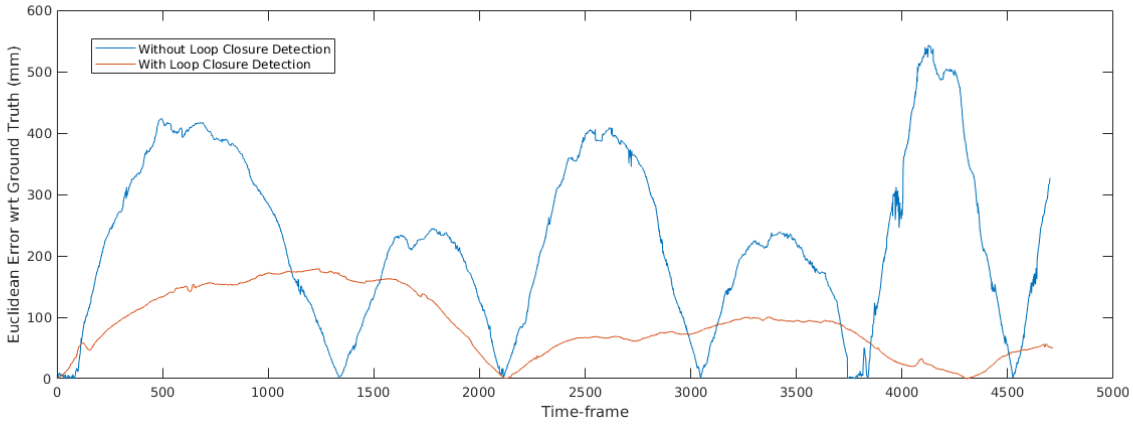
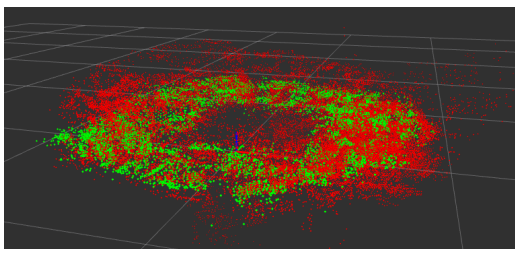


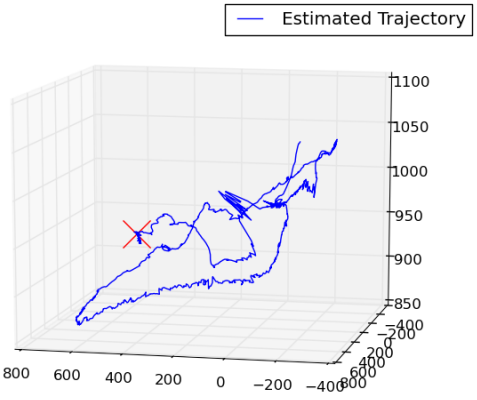
Figure 6.6: Comparison of Euclidean Errors at Each Frame for the Algorithm With and Without Loop Closure Detection

It can be seen that the error in the final trajectory after loop closure optimisation is much less (Mean: 52.1 mm) compared to that without loop closure detection (Mean: 235.4 mm), reducing the average error by **77.8%**. Figure 6.7 shows the maps estimated by the algorithm with and without graph optimisation using loop closures.



Green: With Loop Closure Detection and Graph Optimisation
Red: Without Graph Optimisation

Figure 6.7: Complete Map Estimation for Dataset 2



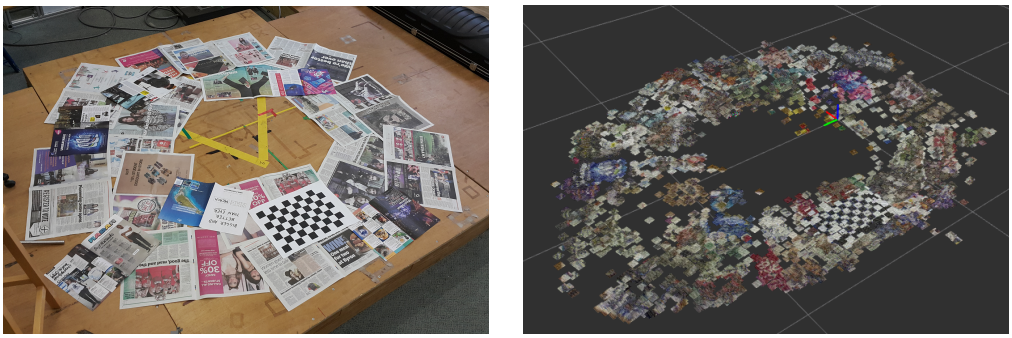
(Red 'X' marks the starting point)

Figure 6.8: Wrong Trajectory Estimation Due to False Loop Closure Detection

The path had 16 detectable loop closures, out of which 12 were successfully detected by the algorithm. The implemented algorithm (Algorithm 3) is robust against false loop closure detection (Section 6.2), and hence did not cause irrecoverable errors in the trajectory and map estimates. Figure 6.8 shows how the trajectory estimate for the same dataset (Dataset 2) is completely ruined due to just one false loop closure detection.

[1]The reason for the errors going down to zero and up again (Figure 6.6) is because of the change in sign of the error; as the estimated path crosses the ground truth, the direction of error changes, resulting in the Euclidean error reducing to zero as the paths intersect, and rising again as they deviate.

The final estimated map and the actual scene are shown in Figure 6.9.



(a) Actual Scene (b) Estimated Map

Figure 6.9: Actual Scene and Final Map Estimate

Loop Closure Detection: Comparison with FAB-MAP

The modifications made to the FAB-MAP algorithm for avoiding false detections (explained in Section 5.2.1) are justified in this section. The modifications were tested on Dataset 2. The original FAB-MAP algorithm detected 4034 false positives out of the 4721 frames (85.45% false positives). This is because each image is added to the Test-Data as soon as it is observed, leading to too many similar images to compare with. False loop closures are detected because subsequent images are similar and also because FAB-MAP works only on datasets that have object variety across the images. As explained previously, the datasets created for the project provided little in terms of object variety.

The detection performance was tested first by making the images in the Test-Data of FAB-MAP sparse. This was done by ensuring that only every $\mathbf{200}^{\text{th}}$ image is added to the Test-Data. For the sequence in Dataset 2, this sparse Test-Data contained just 8 images for one round around the trajectory (instead of over 1600, in case of the actual FAB-MAP algorithm). Since the dataset consists of three rounds, the drone sees the images for the first time in its first round, and should identify them again in both the second and third rounds. Therefore the total number of true positives is 16.

Another modification made to the FAB-MAP algorithm is the *Repeated Match* test. The algorithm ensures that FAB-MAP matches at least **10** subsequent images to the same image in the now-sparse Test-Data for it to be a possible loop closure.

The final modification made to the FAB-MAP algorithm is verifying the detection using the *Optical Flow Check*. This ensures that the positions of all (or most of) the features in the two images can be explained using a single camera motion.

The detections made by these modifications on Dataset 2 are shown in Figure 6.10. It shows how the number of false positives falls to zero with each modification at the cost of losing some true detections. The 'sparse' modification was able to detect all the loop closures correctly, but it detected too many false positives (39.5%, which is almost 2000 false detections) to be a good loop closure detection algorithm for the vSLAM project. The final algorithm with the addition of the 'Repeated Match' and 'Optical Flow Check' was able to detect **12** of the 16 loop closures, and successfully avoided all false detections.

The 'Optical Flow Check' alone could be used as a loop closure detection method, however it is computationally more costly. Therefore, it has to be used only as final verification for the loop closure detection algorithm. By ensuring 'Repeated Matches', only highly probable loop closures have to be verified using the 'Optical Flow Check'. Moreover, the 'Repeated Matches' check hardly costs any extra computation as it simply

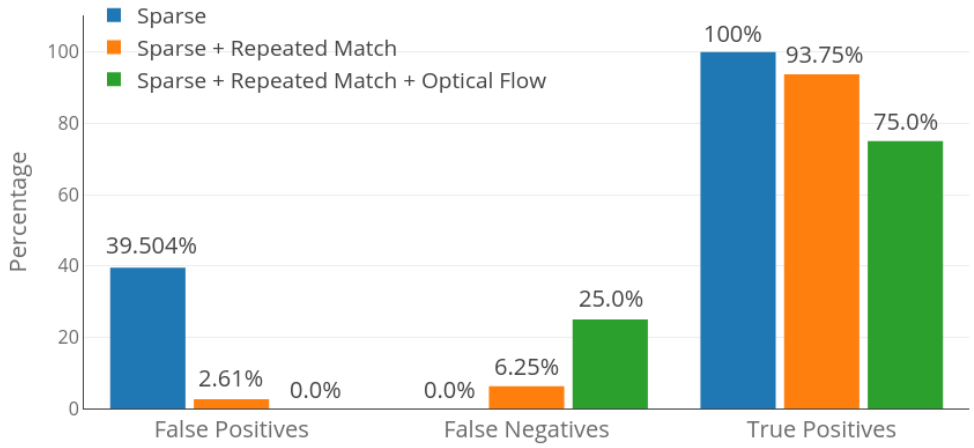


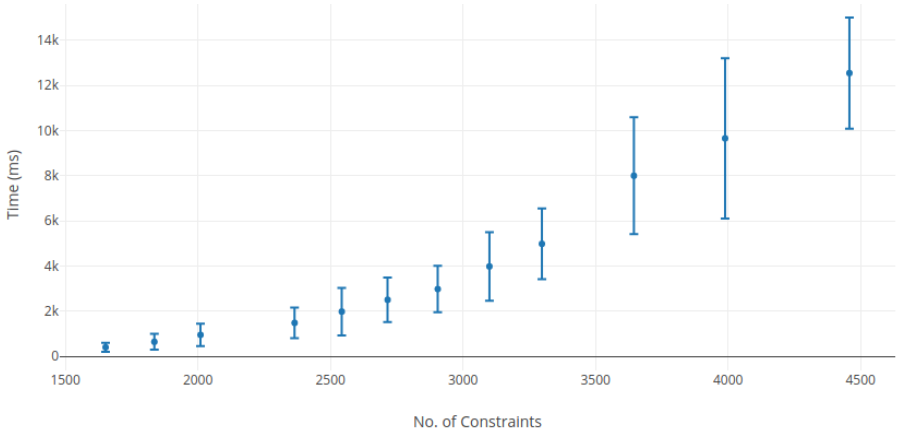
Figure 6.10: Detections Made by the Loop Closure Detection Modifications

checks if FAB-MAP gives the ID of the same image (from the Test-Data) consecutively.

Processing Time and Computational Performance[2]

Optimisation Step

The average time required for performing vSLAM using the developed algorithm on a dataset without loop closure detection and graph optimisation was found to be 1.2 times more than the actual running time of the image sequence. The time required for the graph optimisation step increases with the number of constraints in the graph, hence making the total run time of the algorithm dependent on the number of loop closure detections.



(The error bar indicates the standard deviation)

Figure 6.11: Mean Graph Optimisation Time for Increasing Number of Constraints

The algorithm was run five times on Dataset 2 and the average time at each optimisation step noted. Figure 6.11 shows the mean time taken for the graph optimisation step and the corresponding number of constraints being optimised. As expected, the time required for graph optimisation can be seen to be increasing with the increase in the number of constraints. The variation can be seen as being consistent with the linear complexity

[2]All computations were done on an i7-5500U CPU @ 2.40GHz, 6GB RAM laptop computer running Ubuntu 14.04 (mentioned in Section 5.1.1).

of the graph-based SLAM problem mentioned in literature (complexity increases linearly with number of constraints) [36].

Computation Time for Loop Closure Detection

The 'Optical Flow Check' step was found to take approximately 0.048s. This is in addition to the time taken by FAB-MAP to identify a matching image from the Test-Data. However, due to the sparsity of the data in the modified loop detection, the time required for finding a match from the Test-Data is reduced. The 'Repeated Matches' test, as mentioned previously, does not require any extra computation, but simply acts as a filter to send only the more probable loop closures to the 'Optical Flow Check'.

However, it is undeniable that the speed of the algorithm is affected due to the more extensive checking for true loop closures. But, since the performance of the vSLAM algorithm can be drastically affected by false loop closure detections as shown previously, this price had to be paid. Moreover, detecting larger number of (false) loop closures would lead to the triggering of the graph optimisation step more often – with more number of constraints – which would result in even more loss of speed and efficiency.

The modified algorithm was able to provide much better results due to the modifications made. Also, as the entire project was developed with the intention of estimating a trajectory and map from a sequence of images *offline*, and not in real-time, time required for solving the problem may be considered to be less important – a 'soft constraint'.

Chapter 7

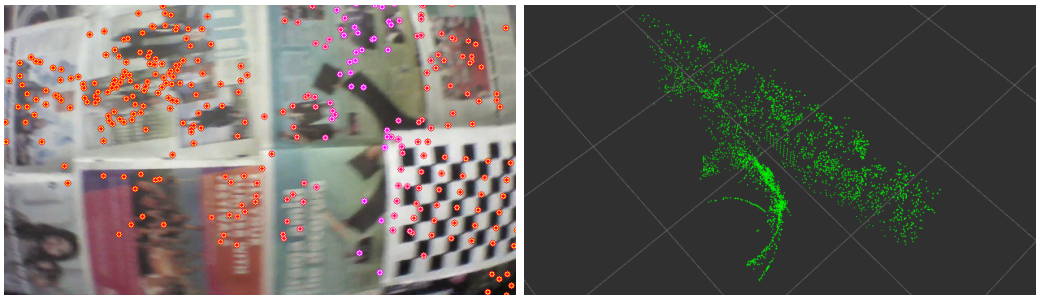
Project Analysis - Failure Cases and Performance Evaluation

The vSLAM algorithm developed was shown to be able to estimate a map and trajectory for a path as long as 24 m with a total average error less than 53 mm, as long as the scene is highly textured and has a few distinguishable loop closures. The two mapping methods that were developed provided reasonably good representations of the environment in terms of visualisation and landmark positions (for loop closure detection). The modifications made to the FAB-MAP algorithm reduced the number of false detections to zero, bringing the error in trajectory estimation down by about 78%. As computation time is not a highly demanding constraint for an offline mapping technique such as graph-based SLAM, the slight decrease in the speed of the algorithm is not considered to be a major drawback. All these features make the developed vSLAM algorithm a reasonably good attempt at solving the graph-based SLAM problem using only a monocular camera.

However, the imperfections and failure cases that were noticed throughout the development and testing of the algorithm cannot be overlooked. An important factor that was initially observed to result in the failure of the algorithm was optimising the graph with wrong loop closures (Figure 6.8). Majority of the time for the project development had to be devoted to come up with a technique to avoid detecting false loop closures without slowing down the code tremendously. As discussed, the modifications implemented in the project were able to overcome this problem and give reasonably good map and trajectory estimations, while not causing drastic delays in the overall run-time of the algorithm.

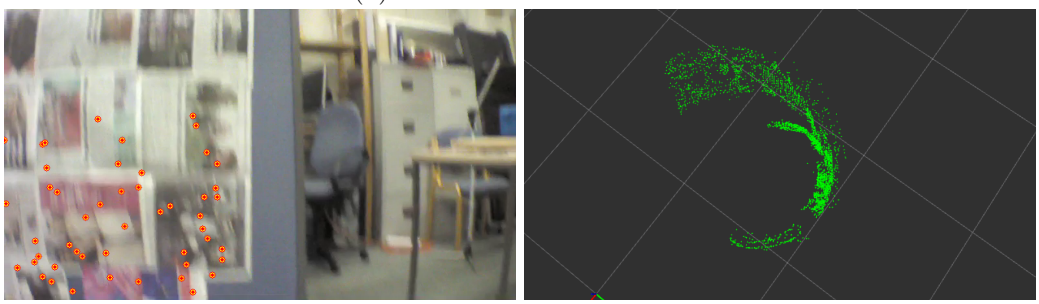
Besides being sensitive to the accuracy of the constraints in the graph, the developed vSLAM algorithm was found to be very much dependent on the performance of the visual odometry. Therefore, most of the problems faced during the implementation of the vSLAM algorithm were due to VO failure. Some of the issues that came up in the development of the vSLAM algorithm due to failure of the VO algorithm have been shown in Figure 7.1.

As explained in Section 4.1, the VO algorithm depends on various visual properties of the scene such as texture, illumination and feature variety, in addition to camera motion stability and camera properties such as resolution and frame rate. The main reason for the extreme sensitivity to the scene was found to be the indirect (feature-based), sparse approach that had been adopted for visual odometry. Such approaches depend on the number of distinguishable features in the scene that can be detected and matched across images. This requires small camera distance between consecutive frames, no blurred images, and highly textured scenes. One way to avoid these conditions is to use direct VO methods (Section 4.2), which make use of all pixels in an image, instead of specific keypoints. Such methods have been shown to improve visual odometry considerably [53, 54]. LSD-SLAM [54] is a successful SLAM framework that efficiently uses the direct method for estimating camera trajectory and building a dense map. Due to the higher



(a) Unsteady Camera Motion

(b) Lack of Texture in Scene



(c) Loss of Tracked Features in the Subsequent Image

Right: Map Created; Left: Current Image Frame in VO Algorithm

Figure 7.1: Mapping Failure due to Bad Visual Odometry

Scenes from demo video (Appendix B)

computation required by the direct methods, hybrid frameworks - which combines the advantages of the direct and indirect methods to produce semi-dense maps - have been developed and applied in SLAM. Semi-Direct Visual Odometry [63] is one such framework that has been shown to produce very good pose estimation even when the scene is sparsely textured and the camera motion is extremely erratic. Using such frameworks for obtaining the visual odometry should improve the robustness of the vSLAM algorithm.

Another problem with the developed vSLAM algorithm is the lack of a 'back-up' pose estimation method in the VO algorithm for recovery. If the VO fails for a single frame, it breaks the code and the odometry estimations stop. As the vSLAM algorithm has no other source to create nodes for the graph, it too fails as soon as the VO code breaks. Therefore an improvement that can be made to the vSLAM algorithm for to overcome such failures is to implement a back-up technique to continue the graph construction process even if the VO algorithm fails temporarily. It is practically simpler to implement such a step in the VO algorithm rather than modifying the vSLAM process. One possible recovery method that can be tested is to use the previous pose estimation if the VO fails for a particular frame. Similar techniques have been utilised even for terrain exploration by Mars rovers [81], where the algorithm assumes that the vehicle is staying still until a relatable new image or a loop closure is detected.

However, the most inconvenient factor in applying the algorithm to a different dataset

is probably the parameter tuning required. Besides the numerous parameters of the VO algorithm such as the baseline for triangulation and feature detection parameters, the performance of the vSLAM algorithm can be drastically affected by the definition of the information matrices for the constraints in the graph. As there is generally no hard-and-fast set of rules or formula to decide the information matrices to be used, they have to be tuned empirically. The modifications made to the loop closure detection algorithm also has some parameters that have to be tuned for the dataset, such as the minimum number of matches required by the 'Repeated Matches' test, and the number of frames to skip for creating a sparse Test-Data for FAB-MAP. The dependency of the vSLAM algorithm on such a large number of sensitive parameters makes portability and generalisation to different datasets difficult.

Using direct VO methods would greatly reduce the number of parameters to be tuned for reliable odometry. Also, incorporating additional sensors such as IMU would give more robust motion and pose estimation. Using more sensors would also make the constructed graph more flexible, by making the observations and poses less co-dependent. Future work on the project can include using direct or semi-direct VO algorithms, and incorporating more sensors for better state estimation and mapping.

Chapter 8

Conclusion

A graph-based visual SLAM system was successfully developed and tested for this project. The system was able to estimate the trajectory followed by the robot/camera and create a sparse representation of the scene using just the sequence of images captured by the on-board camera. The implementation of the algorithm and the theoretical background required for understanding it were described.

The performance of the developed vSLAM algorithm on three original datasets were tested and the results analysed. It was found that the performance can be highly influenced by the texture in the scene and the motion of the camera. The results also proved that drift in depth estimation is more prominent than in the lateral directions. It was shown that the vSLAM system performs well for trajectories of up to 24 metres if the scene is well-textured and the motion of the camera is steady, with a total average error of less than 5.3 cm.

A novel approach to filter out false loop closure detections was described and tested. It was built on the standard FAB-MAP loop detection framework, by incorporating a few extra steps to validate the detection. Comparing the performance of the modified algorithm with that of FAB-MAP on the same dataset, it was shown how each additional step improved the filtering out of false detections. The final loop detection algorithm detected zero false loop closures while correctly detecting 75% of the true positives. Optimising the graph using true loop closures was shown to reduce the error in trajectory estimation by about 78%.

The computational aspects of the vSLAM algorithm in terms of speed and efficiency were discussed and justified. Finally, the drawbacks and shortcomings of the algorithm were analysed and the effect of visual odometry errors on its performance described. Possible modifications to improve the robustness of the system and reduce its sensitivity to the scene texture and camera motion were also suggested.

References

[1] Giorgio Grisetti et al. "A tutorial on graph-based SLAM". In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[2] Andrew J Davison. "Real-time simultaneous localisation and mapping with a single camera". In: *null*. IEEE. 2003, p. 1403.

[3] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. "An overview to visual odometry and visual SLAM: Applications to mobile robotics". In: *Intelligent Industrial Systems* 1.4 (2015), pp. 289–311.

[4] Kasra Khosoussi, Shoudong Huang, and Gamini Dissanayake. "Good, bad and ugly graphs for SLAM". In: *RSS Workshop on the problem of mobile sensors*. 2015.

[5] Mark Cummins and Paul Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance". In: *The International Journal of Robotics Research* 27.6 (2008), pp. 647–665.

[6] Mark Cummins and Paul Newman. "Highly scalable appearance-only SLAM-FAB-MAP 2.0." In: *Robotics: Science and Systems*. Vol. 5. Seattle, USA. 2009, p. 17.

[7] Nils J Nilsson. *A mobile automaton: An application of artificial intelligence techniques*. Tech. rep. SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, 1969.

[8] Raja Chatila. "Robot Mapping: An Introduction". In: *Robotics and Cognitive Approaches to Spatial Mapping*. Ed. by Margaret E. Jefferies and Wai-Kiang Yeap. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 9–12. ISBN: 978-3-540-75388-9. DOI: 10.1007/978-3-540-75388-9_2. URL: https://doi.org/10.1007/978-3-540-75388-9_2.

[9] Sebestian Thrun. "Robotics and cognitive approaches to spacial mapping". In: *Springer Tracts in Advanced Robotics* 38 (2008), pp. 13–41.

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, 2005.

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. "Robot Motion". In: *Probabilistic Robotics*. 2nd ed. MIT press, 2005, pp. 91–120.

[12] Randall Smith, Matthew Self, and Peter Cheeseman. "Estimating uncertain spatial relationships in robotics". In: *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[13] Michael Montemerlo et al. "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai*. 2002, pp. 593–598.

[14] Dirk Hahnel et al. "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements". In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 1. IEEE. 2003, pp. 206–211.

[15] Jose A Castellanos et al. “The SPmap: A probabilistic framework for simultaneous localization and map building”. In: *IEEE Transactions on Robotics and Automation* 15.5 (1999), pp. 948–952.

[16] Edwin Olson, John Leonard, and Seth Teller. “Fast iterative alignment of pose graphs with poor initial estimates”. In: *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE. 2006, pp. 2262–2269.

[17] Frank Dellaert and Michael Kaess. “Square Root SAM: Simultaneous localization and mapping via square root information smoothing”. In: *The International Journal of Robotics Research* 25.12 (2006), pp. 1181–1203.

[18] Sebastian Thrun and Michael Montemerlo. “The graph SLAM algorithm with applications to large-scale mapping of urban structures”. In: *The International Journal of Robotics Research* 25.5-6 (2006), pp. 403–429.

[19] Jan Helge Klussendorff et al. “Graph-based visual SLAM and visual odometry using an RGB-D camera”. In: *Robot Motion and Control (RoMoCo), 2013 9th Workshop on*. IEEE. 2013, pp. 288–293.

[20] Peter Henry et al. “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments”. In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[21] Johann Borenstein, Liqiang Feng, and HR Everett. “Landmark Navigation”. In: *Navigating mobile robots: Systems and techniques*. AK Peters, Ltd., 1996.

[22] Sami Atiya and Gregory D Hager. “Real-time vision-based robot localization”. In: *IEEE Transactions on Robotics and Automation* 9.6 (1993), pp. 785–800.

[23] Paul Newman and Kin Ho. “SLAM-loop closing with visually salient features”. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, pp. 635–642.

[24] Kurt Konolige et al. “Efficient sparse pose adjustment for 2D mapping”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 22–29.

[25] Feng Lu and Evangelos Milios. “Globally consistent range scan alignment for environment mapping”. In: *Autonomous robots* 4.4 (1997), pp. 333–349.

[26] Wolfgang Hess et al. “Real-time loop closure in 2D LIDAR SLAM”. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 1271–1278.

[27] Ethan Eade and Tom Drummond. “Unified Loop Closing and Recovery for Real Time Monocular SLAM.” In: *BMVC*. Vol. 13. 2008, p. 136.

[28] Siyang Song et al. “A precise and real-time loop-closure detection for SLAM using the RSOM tree”. In: *International Journal of Advanced Robotic Systems* 12.6 (2015), p. 73.

[29] Raúl Mur-Artal and Juan D Tardós. “Fast relocalisation and loop closing in keyframe-based SLAM”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 846–853.

[30] Brian Williams et al. “An image-to-map loop closing method for monocular SLAM”. In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE. 2008, pp. 2053–2059.

[31] Laura A Clemente et al. “Mapping Large Loops with a Single Hand-Held Camera.” In: *Robotics: Science and Systems*. Vol. 2. 2. 2007.

[32] Christoph Hertzberg. "A framework for sparse, non-linear least squares problems on manifolds". In: *UNIVERSITÄT BREMEN*. Citeseer. 2008.

[33] Rainer Kümmerle et al. "g2o: A general framework for graph optimization". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3607–3613.

[34] Edwin Olson. "Recognizing places using spectrally clustered local matches". In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1157–1172.

[35] Giorgio Grisetti et al. "A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent." In: *Robotics: Science and Systems*. 2007, pp. 27–30.

[36] Hauke Strasdat, Jos'e MM Montiel, and Andrew J Davison. "Visual SLAM: why filter?" In: *Image and Vision Computing* 30.2 (2012), pp. 65–77.

[37] P Pinies et al. "CI-Graph SLAM for 3D reconstruction of large and complex environments using a multicamera system". In: *J. of Field Robotics* 27.5 (2010), pp. 561–586.

[38] Cyril Stachniss. *Robot Mapping - WS 2013/14*. 2013. URL: `http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/` (visited on 07/07/2017).

[39] Giorgio Grisetti. "Notes on Least-Squares and SLAM DRAFT". In: (2014).

[40] Thomas B Schön and Fredrik Lindsten. "Manipulating the multivariate gaussian density". In: *Division of Automatic Control, Linköping University, Sweden, Tech. Rep* (2011).

[41] Sebastian Thrun and John J Leonard. "Simultaneous localization and mapping". In: *Springer handbook of robotics*. Springer, 2008, pp. 871–889.

[42] Hordur Johannsson et al. "Temporally scalable visual SLAM using a reduced pose graph". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 54–61.

[43] Ian Mahon et al. "Efficient view-based SLAM using visual loop closures". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1002–1014.

[44] Sunhyo Kim and Se-Young Oh. "SLAM in indoor environments using omni-directional vertical and horizontal line features". In: *Journal of Intelligent & Robotic Systems* 51.1 (2008), pp. 31–43.

[45] Soonhac Hong and Cang Ye. "A pose graph based visual SLAM algorithm for robot pose estimation". In: *World Automation Congress (WAC), 2014*. IEEE. 2014, pp. 917–922.

[46] Peter Henry et al. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[47] Davide Scaramuzza and Friedrich Fraundorfer. "Visual odometry [tutorial]". In: *IEEE robotics & automation magazine* 18.4 (2011), pp. 80–92.

[48] Andrew W Fitzgibbon and Andrew Zisserman. "Automatic camera recovery for closed or open image sequences". In: *European conference on computer vision*. Springer. 1998, pp. 311–326.

[49] D Charnley et al. "The DROID 3D Vision System-algorithms for geometric integration". In: *Plessey Research, Roke Manor, Technical Note* 72.88 (1988), N488U.

[50] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[51] Mark Pupilli and Andrew Calway. “Real-Time Camera Tracking Using a Particle Filter.” In: *BMVC*. 2005.

[52] Georg Klein and David Murray. “Parallel tracking and mapping on a camera phone”. In: *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*. IEEE. 2009, pp. 83–86.

[53] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. “DTAM: Dense tracking and mapping in real-time”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2320–2327.

[54] Jakob Engel, Thomas Schöps, and Daniel Cremers. “LSD-SLAM: Large-scale direct monocular SLAM”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 834–849.

[55] Thomas Whelan et al. “ElasticFusion: Dense SLAM without a pose graph”. In: Robotics: Science and Systems. 2015.

[56] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[57] Daniel M Helmick et al. “Path following using visual odometry for a mars rover in high-slip environments”. In: *Aerospace Conference, 2004. Proceedings. 2004 IEEE*. Vol. 2. IEEE. 2004, pp. 772–789.

[58] Heng Zhang, Yanli Liu, and Jindong Tan. “Loop closing detection in RGB-D SLAM combining appearance and geometric constraints”. In: *Sensors* 15.6 (2015), pp. 14639–14660.

[59] Xuanpeng Li and Rachid Belaroussi. “Semi-Dense 3D Semantic Mapping from Monocular SLAM”. In: *arXiv preprint arXiv:1611.04144* (2016).

[60] Ethan Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *Computer Vision (ICCV), 2011 IEEE international conference on*. IEEE. 2011, pp. 2564–2571.

[61] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. “ORB-SLAM: a versatile and accurate monocular SLAM system”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163.

[62] Kudan Computer Vision. *Different Types of Visual SLAM Systems*. 2017. URL: `https://www.kudan.eu/kudan-news/different-types-visual-slam-systems/` (visited on 08/25/2017).

[63] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. “SVO: Fast semi-direct monocular visual odometry”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 15–22.

[64] Michael Bosse and Paul Newman. “JLST. Slam in large-scale cyclic environments using the atlas framework”. In: *Internatioonal Journal of Robotics Research* (2004).

[65] Kin Leong Ho and Paul Newman. “Loop closure detection in SLAM by combining visual and spatial appearance”. In: *Robotics and Autonomous Systems* 54.9 (2006), pp. 740–749.

[66] Brian Williams et al. “A comparison of loop closing techniques in monocular SLAM”. In: *Robotics and Autonomous Systems* 57.12 (2009), pp. 1188–1197.

[67] Josef Sivic and Andrew Zisserman. “Video Google: A text retrieval approach to object matching in videos”. In: *null*. IEEE. 2003, p. 1470.

[68] Jana Košecká, Fayin Li, and Xialong Yang. “Global localization and relative positioning based on scale-invariant keypoints”. In: *Robotics and Autonomous Systems* 52.1 (2005), pp. 27–38.

[69] Dorian Galvez-Lopez and Juan D Tardos. "Real-time loop detection with bags of binary words". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 51–58.

[70] Adrien Angeli et al. "Fast and incremental method for loop-closure detection using bags of visual words". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1027–1037.

[71] Dorian Gálvez-López and Juan D Tardos. "Bags of binary words for fast place recognition in image sequences". In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197.

[72] Jacobo Jiménez Lugo and Andreas Zell. "Framework for autonomous on-board navigation with the AR. Drone". In: *Journal of Intelligent & Robotic Systems* 73.1-4 (2014), pp. 401–412.

[73] Oqus. *Qualysis*. 2017. URL: `http://www.qualisys.com/cameras/oqus/`.

[74] OpenCV. *Camera Calibration and 3D Reconstruction*. 2017. URL: `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`.

[75] Constance S Royden and Kathleen D Moore. "Use of speed cues in the detection of moving objects by moving observers". In: *Vision research* 59 (2012), pp. 17–24.

[76] Daniel Scharstein and Richard Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *International journal of computer vision* 47.1-3 (2002), pp. 7–42.

[77] Olof Enqvist. *Correspondence Problems in Geometric Vision*. Centre for Mathematical Sciences, Mathematics, Lund University, 2009.

[78] Mathieu Labbe and François Michaud. "Online global loop closure detection for large-scale multi-session graph-based slam". In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE. 2014, pp. 2661–2666.

[79] David Eigen and Rob Fergus. "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2650–2658.

[80] M Zeeshan Zia, Michael Stark, and Konrad Schindler. "Towards scene understanding with detailed 3d object representations". In: *International Journal of Computer Vision* 112.2 (2015), pp. 188–203.

[81] Mark Maimone, Yang Cheng, and Larry Matthies. "Two years of visual odometry on the mars exploration rovers". In: *Journal of Field Robotics* 24.3 (2007), pp. 169–186.

[82] John M Lee. "Smooth manifolds". In: *Introduction to Smooth Manifolds*. Springer, 2003, pp. 1–29.

[83] Ermano Arruda, Joao M Teixeira, and Saif Sidhik. *STAM - Simple Tracking and Mapping*. 2017. URL: `https://github.com/eaa3/STAM/tree/graphslam_mod`.

The references have been prepared using the Modern Language Association of America (MLA) format, *MLA Handbook, Eighth Edition*.

Appendix A

Graph-Based SLAM in Practice

Solving $\mathbf{H}\Delta\mathbf{x}^* = -\mathbf{b}$

Theoretically, this can be solved by matrix inversion: $\Delta\mathbf{x}^* = -\mathbf{H}^{-1}\mathbf{b}$. However, this is practically very difficult to do due to its high calculation complexity (cubic complexity) [38]. In practice, these equations are solved using efficient methods such as Cholesky factorisation, QR decomposition or iterative methods such as conjugate gradients [38, 39].

Without going into the derivation, the steps involved in the Cholesky factorisation for solving a system given by $\mathbf{Hx} = \mathbf{b}$ can be summarised as follows: If $\mathbf{H}$ is a symmetric positive definite matrix, the Cholesky decomposition of $\mathbf{H}$ leads to

$$\mathbf{H} = \mathbf{L}\mathbf{L}^T$$

where $\mathbf{L}$ is a lower triangular matrix with real and positive diagonal entries, and is called the Cholesky factor of $\mathbf{H}$.

This is used in the equation $\mathbf{Ly} = \mathbf{b}$ to solve for $\mathbf{y}$ by forward-substitution. Then, this solution for $\mathbf{y}$ is used to solve $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by back-substitution. This way the solution for $\mathbf{x}$ is obtained without using matrix inversion, thereby reducing the computational complexity.

Error Minimisation Using Levenberg-Marquardt

The Gauss-Newton algorithm (Algorithm 1) is not guaranteed to converge for all cases, and its convergence depends on the initial guess, smoothness of the error function, potential singularities etc. It might also happen that one iteration of Gauss-Newton results in a new solution that is worse than the initial one.

One way of overcoming this problem is by using the Levenberg-Marquardt (LM) algorithm, which is a damped version of the Gauss-Newton. It has 'backup-restore' capability to recover from wrong steps. LM solves *damped* version of the system $\mathbf{H}\Delta\mathbf{x}^* = -\mathbf{b}$, at every iteration. If λ is the damping factor, the damped form of the system can be formulated as:

$$(\mathbf{H} + \lambda\mathbf{I})\Delta\mathbf{x}^* = \mathbf{b}$$

The steps in an iteration of the LM algorithm is exactly the same as those in the Gauss-Newton algorithm, except for the final adjustment step. At the end of each iteration, the LM algorithm computes the new error of the system using the new solution obtained. If the error is less than the error in the previous iteration, the damping factor λ is reduced. On the other hand, if the error is larger than the previous error, the value of λ is increased

and the previous solution is restored. A new solution using the new value of λ is obtained, and the error calculated. This is repeated a pre-defined number of times before fixing on a solution for that iteration. An abstract implementation of the LM method is shown in Algorithm 4.

Algorithm 4 Levenberg-Marquardt Error Minimisation Algorithm

Require: Initial Guess $\breve{\mathbf{x}}$; and measurements $\mathcal{C} = \{\langle z_k(\cdot), \Omega_k \rangle\}$
Ensure: $\mathbf{x}^*$ = new solution

$\boldsymbol{F}_{new} \leftarrow \breve{\boldsymbol{F}}$ ▷ *current error*

// backup the solution

$\mathbf{x}_{\text{backup}} \leftarrow \breve{\mathbf{x}}$

$\lambda \leftarrow \text{computeInitialLambda}(\mathcal{C}, \breve{\mathbf{x}})$

// iterate till no substantial improvements attained

repeat

  $\breve{\boldsymbol{F}} \leftarrow \boldsymbol{F}_{new}$

  $\mathbf{b} \leftarrow 0 \quad \mathbf{H} \leftarrow 0$

  for all $k = 1...K$ **do**

    $\breve{z}_k \leftarrow h_k(\breve{\mathbf{x}})$

    $\mathbf{e}_k \leftarrow \hat{z}_k - z_k$

    $\mathbf{J}_k \leftarrow \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$

  // computing contribution of the measurement in the linear system

    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{J}_k; \quad \mathbf{b}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{e}_k$

  // accumulating contributions to build overall system

    $\mathbf{H}$ += $\mathbf{H}_k$; $\mathbf{b}$ += $\mathbf{b}_k$

  end for

  // adjust damping factor and perform backup/restore action

  $t \leftarrow 0$ ▷ *number of iterations the damping factor has been adjusted*

  repeat

  // solve linear equation to get new increment

    $\Delta\mathbf{x} \leftarrow \text{solve}((\mathbf{H} + \lambda\mathbf{I})\Delta\mathbf{x} = -\mathbf{b})$

  // update state

    $\breve{\mathbf{x}}$ += $\Delta\mathbf{x}$

    $\boldsymbol{F}_{new} \leftarrow \boldsymbol{F}(\breve{\mathbf{x}})$ ▷ *new error*

    if $\boldsymbol{F}_{new} < \breve{\boldsymbol{F}}$ **then** ▷ *good step; accept solution and update temporaries*

      $\lambda \leftarrow \lambda/2$

      $\mathbf{x}_{\text{backup}} \leftarrow \breve{\mathbf{x}}$

      $t \leftarrow -1$

    else ▷ *bad step; revert to previous solution*

      $\lambda \leftarrow \lambda * 4$

      $\breve{\mathbf{x}} \leftarrow \mathbf{x}_{\text{backup}}$

      t += 1

    end if

  until $t < t_{\max} \wedge t > 0$

until $\breve{\boldsymbol{F}} - \boldsymbol{F}_{new} < \epsilon \wedge t < t_{\max}$

return $\breve{\mathbf{x}}$

In general, the LM algorithm always converges to a minimum, but it might not be the global one. However, due to the backup-restore nature of the algorithm, it is more likely

to give true estimates compared to Gauss-Newton minimisation.

Operations on Smooth Manifolds

The assumption made in Section 3.2 while solving the state estimation problem was that all the state variables of $\mathbf{x}$ were in one space (Euclidean). The issue when applying the least-squares approach in practice is the problem of *smooth manifolds* [82]. If all the state variables of $\mathbf{x}$ had been in one space (say Euclidean), this would not have been a concern. In robotics however, this is often not the case, since angles and rotations are involved. Summing a triplet of Euler angles is not always meaningful, and might lead to singular configurations. Therefore, additions and subtractions do not work the same way for rotations. In fact, rotations are not Euclidean, although they admit local Euclidean mappings, which makes them *smooth manifolds*.

A manifold is a space that might not be Euclidean globally, but behaves like a Euclidean space at a local scale [82]. To understand the contents of this section, consider the space of 3D rotations. Rotations in 3D can be parametrised with a rotation matrix $\mathbf{R}$. A rotation matrix, however has 9 numbers to represent 3 angles. Using rotation matrices as parameters in the optimization mechanism (Algorithms 1 and 2) may lead to non-valid solutions, since the orthogonality constraint is not enforced. Thus, it seems there is no alternative than carrying on the optimization by using a minimal representation, for instance Euler angles. Unfortunately, as stated above, Euler angles suffer from singularities. However, rotations are a manifold, and they admit a local parametrisation which is homeomorphic to the vector space $\mathbb{R}^n$. To understand this statement, consider a generic rotation $\mathbf{R}(\rho, \theta, \psi)$. A mapping between rotation matrices and Euler angles can be easily defined as follows:

$$\mathbf{u} = \text{toVector}(\mathbf{R})$$

$$\mathbf{R} = \text{toMatrix}(\mathbf{u})$$

where $\mathbf{u} = (\phi\ \theta\ \psi)^T$ is the vector containing the 3 Euler angles.

Clearly, $\text{toMatrix}(\text{toVector}(\mathbf{R})) = \mathbf{R}$ and $\text{toVector}(\text{toMatrix}(\mathbf{u})) = \mathbf{u}$. Around the origin of u, the Euler angles are away from singularities and the Euclidean distance measured between two orientations $\mathbf{u}_1$ and $\mathbf{u}_2$ around the origin are closely related to the true distance of their orientations. However, when close to a singularity, a small difference in orientation might lead to an abrupt change in the local parameters $\mathbf{u}$, with potentially catastrophic consequences on the optimization, as it strongly relies on the smoothness of the problem.

However, given any arbitrary initial rotation $\mathbf{R}_0$, a mapping to Euler angles can be computed, one which is locally away from singularities. In the domain of the rotations, this can be easily done by setting the reference frame so that $\text{toVector}(\mathbf{R}_0) = 0$ is the origin,

$$\mathbf{u} = \mathbf{R} \boxminus \mathbf{R}_0 = \text{toVector}(\mathbf{R}_0^{-1}\mathbf{R}) \tag{A.1}$$

$$\mathbf{R} = \mathbf{R}_0 \boxplus \mathbf{u} = \mathbf{R}_0 \cdot \text{toMatrix}(\mathbf{u}) \tag{A.2}$$

Equation A.1 returns the Euler angles of the rotation that moves $\mathbf{R}_0$ to $\mathbf{R}$, in the reference frame of $\mathbf{R}_0$. It computes the coordinates in a local map centred around $\mathbf{R}_0$. If the two rotations have a small difference, $\mathbf{u}$ will be small, independent of whether $\mathbf{R}_0$ is close to a singularity or not. Similarly, Equation A.2 computes a new rotation from $\mathbf{R}_0$, by applying a local perturbation $\mathbf{u}$. It is obvious that $\mathbf{R}_0 \boxplus (\mathbf{R} \boxminus \mathbf{R}_0) = \mathbf{R}$ and $(\mathbf{R}_0 \boxplus \mathbf{u}) \boxminus \mathbf{R}_0 = u$. The operators $\boxplus$ and $\boxminus$ are powerful operators that convert a global difference in the manifold space to a local perturbation, and vice versa.

Having explained them in the rotation domain, these operators can now be used to solve the optimisation problem by assuming that our state space (with position and orientation of the robot) is a smooth manifold. Consider the values in this space are parametrised redundantly by a certain variable $\mathbf{X}$, and minimally by a vector $\mathbf{x}$. So the observation model has the forms $h_k(\mathbf{X})$ or $h_k(\mathbf{x})$. If $\breve{\mathbf{X}}$ is the current/initial estimate, the perturbation of the predicted observation, under a small variation of the state variables can be obtained using Taylor expansion as follows [39]:

$$\begin{aligned} h_k(\breve{\mathbf{X}} \boxplus \Delta\mathbf{x}) &\simeq h_k(\breve{\mathbf{X}}) + \underbrace{\left.\frac{\partial h_k(\breve{\mathbf{X}} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0}}_{\tilde{\mathbf{J}}_k} \cdot \Delta\mathbf{x} \\ &= h_k(\breve{\mathbf{X}}) + \tilde{\mathbf{J}}_k \Delta\mathbf{x} \end{aligned} \tag{A.3}$$

This equation is very similar to Equation 3.5. However, this expansion gives the change of the parameters as a function of the variation of the increments, which are applied to the current linearisation point $\breve{\mathbf{X}}$ through the $\boxplus$ operator. Therefore, once a solution for $\Delta\mathbf{x}^*$ is found, the perturbation to the current estimate has to be applied using the $\boxplus$ operator.

$$\breve{\mathbf{X}} = \breve{\mathbf{X}} \boxplus \Delta\mathbf{x}^*$$

Similarly, all the minus $(-)$ signs will have to be replaced with $\boxminus$, and all the plus $(+)$ signs with $\boxplus$, whenever the state vectors come into play in the optimisation problem summarised in Algorithm 1. This also necessitates the computation of Jacobians over the manifold, and projecting the Jacobians of the error function to the manifold space.

Further mathematical details are beyond the scope of this report, and the reader is invited to refer to [39] and [1] for more in-depth description and derivation of the same least-squares state estimation problem, but for smooth manifolds.

Appendix B

Code Repository and Demo Video

Source Code

The project source code is available at:

`https://git-teaching.cs.bham.ac.uk/mod-msc-proj-2016/sxs1412`

It was created as a ROS package, and depends on *catkin*. The package requires the visual odometry library, STAM (Simple Tracking and Mapping) [83], based on which the mini-project was developed. It also has various other dependencies (OpenCV, g2o etc.), required for image processing and graph optimisation. The dependencies and requirements are listed in the 'README.md' file in the repository. The file also contains detailed instructions regarding the installation of the package and its dependencies, along with instructions for running its nodes.

External libraries (mentioned in 'README.md') have been used for run-time efficiency and to the make the codes less complicated. The open-source FAB-MAP C++ frame-work [6] has been used and modified for loop closure detection. All other parts of the submitted codes are original.

Demo Video

A video demonstrating the working of the developed algorithm is available at:

`https://www.youtube.com/watch?v=gfz0dfJCWEU`