Federal University of Pernambuco

BSc Computer Science

Graduation Project

A study on SLAM techniques with applications on robot perception

Author:
Ermano Ardiles Arruda

Supervisor:
Veronica Teichrieb
Co-supervisor:
João Paulo Lima

24 de Julho de 2015

Abstract

To simultaneously localise itself and map its surroundings is an essential ability that a robot needs to have in order to perform various tasks in unknown, dangerous or domestic environments. In this work we develop a system capable of building globally consistent maps using a graph-based interpretation of the simultaneous localization and mapping problem. We use an RGBD sensor to extract visual and depth information for loop closure detection and relative transformation estimation. The final system is tested on an openly available benchmark. Finally, experiments are also performed on a mobile robot in a domestic environment, yielding promising positive results.

Resumo

A capacidade de se localizar e mapear o ambiente que o circunda é uma habilidade essencial um robô precisa possuir para realizar diversos tipos de tarefas, sejam elas em ambientes domésticos, desconhecidos ou perigosos para seres humanos. Neste trabalho nós desenvolvemos um sistema capaz de construir mapas globalmente consistentes usando uma interpretação baseada em grafos para o problema de localização e mapeamento simultâneos. Nós usamos um sensor RGBD para extrair informações visuais e de profundidade para detecção e estimativa de transformações relativas de loop closure. O sistema final é testado usando um benchmark disponível na literatura. Finalmente, experimentos usando um robô móvel em um ambiente doméstico foram realizados exibindo resultados promissores.

Acknowledgment

I wanted to thank professor João Paulo Lima for supporting me throughout the whole construction process of this work in this exciting topic. I also wanted to thank all my friends from Voxar Labs for the many advices they gave me. Finally, I wanted to thank professor Veronica Teichrieb for putting me together with all those people.

Contents

List of Figures

1 Introduction

I hear and I forget. I see and I remember. I do and I understand.

Confucius

A classical problem in mobile robotics refers to the ability of a robot to simultaneously localize itself while at the same time map its surroundings, a problem often called Simultaneous Localization and Mapping (SLAM). Already in our daily lives, robots need to solve this problem for performing various tasks, such as vacuum cleaning, lawn mowing, autonomous driving, tour guiding, among other tasks [19, 24, 2].

Indeed, one can think of applications that go beyond the domestic context including space and deep sea exploration, navigation in dangerous environments, and many more. In this work we develop a system capable of building globally consistent maps using a graph-based interpretation of the SLAM problem.

Different types of sensors are commonly used to provide input for SLAM solutions. Classically, laser and sonar sensors have been widely applied for 2D SLAM, providing distance measurements to obstacles nearby, such as walls and other objects. Laser sensors have been also used for 3D SLAM, in which the robot's position now lies in space and the map becomes a reconstruction of the whole 3D scene [10, 24]. Throughout the last decade, a number of researchers began to approach the SLAM problem with mainstream and low cost sensors, such as RGB cameras [25, 6, 8], and RGBD sensors, such as the Microsoft Kinect [7, 14]. This initiative opened up a range of new possibilities in the field, paving the way towards the usage of cheaper hardware for robotic applications.

Nonetheless, although SLAM is a relatively well studied problem in robotics, it is still far from being a closed problem. In order to develop a complete autonomous system, a robot must be able to run for very long periods of time. Therefore, it has to perform mapping, and also manage the generated map according to some particular task being performed. In this context, there is still room for discoveries and alternative interpretations with which SLAM-like formulations might be effectively employed. This application can solve or aid the solution of a wide range of problems in robotics and computational vision. In summary, SLAM as a perceptual mechanism that continuously updates the internal model a certain robot has of the world can also be studied regarding the properties that different exploratory behaviours might yield with respect to performance and quality of the final generated model. This is also related to an open problem in robotics called *Next Best View Problem* [20] which has applications in autonomous exploration and mapping, 3D

reconstruction and even object manipulation.

1.1 Objectives

Given this exciting horizon ahead, our main goal in this work was to create a solid understanding regarding the state of the art solutions for the SLAM problem. And at the same time, we wanted to implement our own approach based on the most recent developments in the area.

1.2 Outline

This work can also be seen as an introductory text on the area and it is organised as follows. In Chapter 2 we start by giving an introduction to the mathematical tools utilised to model the SLAM problem, which include a brief discussion about *manifolds*, *Lie Groups* and the *Maximum Likelihood Method (ML)*. Then, in Chapter 3 we give an overview about some related work on the area. Afterwards, in Chapter 4 we describe in more details the implemented approach, in particular the two main components of a Graph SLAM system so called back-end and front-end, each one responsible for a particular task in the system. In Chapter 5 we examine the experiments performed on a benchmark and in a domestic environment to test the implemented system and discuss their results, giving an understanding of the current status of the implemented approach. Finally, we conclude this work in Chapter 6, also disclosing some future work plans.

2 Mathematical Background

> And everyone must lose his mind, everyone must! The sooner the better! It is essential — I know it.
>
> Yevgeny Zamyatin, We

2.1 Manifolds

Many optimisation procedures implicitly make a set of assumptions about the underlying space they operate, in particular regarding the continuity and linearity of these spaces. This is the case with well-known techniques such as the Kalman filter, particle filters and a number of different approaches designed to work with Euclidean $\mathbb{R}^n$ vector spaces, where n denotes the dimension or number of degrees of freedom of the space. However, it is often the case that these assumptions are not met. As a consequence, in many cases one might get suboptimal results, results that do not make sense and, indeed, the optimisation procedure might even diverge.

In order to solve these problems one has to take into account the true nature of the underlying space, which is commonly not Euclidean, instead, it is a *manifold*. A *manifold* is a space that might not be Euclidean globally, but it behaves like an Euclidean space at a local scale [15]. Any $\mathbb{R}^n$ vector space is a manifold itself. Another trivial example of manifold is a line in $\mathbb{R}^2$ or a plane in $\mathbb{R}^3$, which are manifolds with one and two degrees of freedom, respectively. We can see, however, that these examples are Euclidean both at global and local scale. A more representative example is the surface of the unit circle: we can uniquely identify every point of the circle with a single angle $\theta \in [0, 2\pi[$. Nonetheless, although this space has one degree of freedom, it does not behave like an ordinary $\mathbb{R}^1$ vector space. For example, the difference between two angles cannot be found using simple subtraction of two members of the space, since it is defined to be the shortest angle between them. We could attempt to change the way we represent a member of the circle by using unit vectors instead of a single angle, but this would not solve the problem either: the sum or subtraction of two elements of the manifold would not yield another member of the manifold, i.e. we would break the unit constraint unless we apply some normalization after the sum. These same representations for the unit circle can also be used to represent rotations in $\mathbb{R}^2$, since a single angle $\theta \in [0, 2\pi[$ can also be used for this purpose. Therefore, rotations in $\mathbb{R}^2$ also form a manifold and in this case they are *homeomorphic* to the unit circle.

The meaning behind being "locally Euclidean" is that the manifold is locally *homeomorphic* to $\mathbb{R}^n$.

Definition 1. *Suppose M and N are topological spaces. M is locally homeomorphic to N if for every point $p \in M$ there is an open set $U \subset M$ containing p, for which there exists an open set $\bar{U} \subset N$ and a function $\varphi : U \to \bar{U}$, which is a homeomorphism, i.e. φ is a bijective, continuous map with a continuous inverse.*

The function φ is called a *local coordinate map* and U is called *coordinate domain*. And together they form a *coordinate chart*.

Definition 2. *A pair (U_i, φ_i), such that φ_i is a homeomorphism and $p_i \in U_i$ is called a coordinate chart whose domain contains $p_i \in U_i$.*

It is possible to combine the coordinate charts in order to form a so called *atlas* of M.

Definition 3. *$\mathcal{A} = \{(U_i, \varphi_i)\}$, $\forall i \in \{1, \ldots, n\}$ is called an atlas of M if and only if $U = \bigcup_{i=1}^{n} U_i$.*

In particular, if the transition between any coordinate chart pair is continuous

$$\forall (U_i, \varphi_i), (U_j, \varphi_j) \in \mathcal{A}, \tag{1}$$

then, the combined structure $(M, \mathcal{A})$ is called a *smooth* manifold. It is also possible to combine all charts into a single chart φ that covers the whole manifold, and if the charts are smooth compatible (transitions between them are continuous), then this composite chart will also be. This basically means we can do calculus on such structures, e.g. compute volumes with integrals and curvatures with derivatives. We will not enter in more details about such definitions, but we encourage the reader to refer to [15] for a detailed read on this topic.

In practical terms, the main idea is to create a continuous relationship between the two topological spaces M and N, such that we are able to relate the two spaces and, in a sense, translate in a meaningful way the act of moving inside M to a movement inside N, and vice-versa. The theory behind smooth manifolds together with a few other set of mathematical tools plays an important role on Graph SLAM optimisation, since that is exactly the nature of the state space operated by this procedure. For this reason we will define two useful manifold operators in the following section.

2.2 Manifold Operators

Following [17, 12, 10], we can define some useful operators to work with manifolds. The first operator $\boxplus$ maps small movements in Euclidean space to movements in the manifold M. For δ sufficiently small,

$$\boxplus : M \times \mathbb{R}^n \to M, \tag{2}$$
$$x \boxplus \delta = \varphi_p^{-1}(\varphi_p(x) + \delta). \tag{3}$$

The second operator $\boxminus$ recovers the Euclidean difference between two members of M. This operator is defined as follows, for sufficiently close members $x, y \in M$:

$$\boxminus : M \times M \to \mathbb{R}^n, \tag{4}$$
$$y \boxminus x = \varphi_p(y) - \varphi_p(x). \tag{5}$$

Note the extra notation for φ_p. This is used to express the fact that the coordinate chart φ was chosen such that $\varphi(p) = \mathbf{0}$. It means φ is centred at p. Thus, the notation $\varphi_p(x)$ is used to express this particular choice.

2.3 Groups

A group is a mathematical structure that comprises a set of elements G together with an operation $\boxdot$ which obey three basic axioms:
Closure: If a and b are members of the group, then $c = a \boxdot b$ is a member of the group.
Identity: The group G must contain an identity element I such that $I \boxdot a = a \boxdot I = a$, where $a, I \in G$.
Inverse: Each member a of the group has to have a unique inverse a^{-1} such that $a \boxdot a^{-1} = a^{-1} \boxdot a = I$.

2.3.1 *GL(n)*

A simple example of a group is the general linear group $GL(n)$, which consists of all non-singular square matrices with dimensions $n \times n$. Showing that this is indeed a group is straightforward, first because multiplying two square matrices will always yield another square matrix, so this set is closed under multiplication. Secondly, it is easy to see that the identity element for this set exists, i.e. the identity matrix of dimensions $n \times n$. And finally, these matrices always have unique inverse, because their determinant is always different of zero by construction.

In the context of this work we shall consider three groups of interest with respect to the multiplication operation. The special orthogonal groups $SO(2)$ and $SO(3)$ - subgroups of $GL(n)$ - and the special Euclidean group $SE(3)$, which is a subgroup of $GL(n+1)$. We will see how they actually manifest themselves in practise and useful ways of representing them for computation purposes.

2.3.2 *SO(n)*

The special orthogonal group is defined as the set of all matrices $\mathbf{R}$ with $det(\mathbf{R}) = 1$ that preserves the inner product. In other words,

$$SO(n) = \{\mathbf{R} \in GL(n) | \mathbf{R}^{\mathrm{T}}\mathbf{R} = I \wedge det(\mathbf{R}) = 1\}. \tag{6}$$

In the context of this work, we shall be mainly concerned with $SO(2)$ and $SO(3)$, which are the groups of rotations in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively.

2.3.3 *SE(3)*

We might not be interested only in rotating but also in moving some rigid body in space. The $SE(3)$ group comprises the set of all rigid body motions, i.e. a rotation followed by a translation, that can be applied to some object. This group can be obtained via the Cartesian product between the $SO(3)$ and $\mathbb{R}^3$, and as such it is also a manifold [12, 17]. Note that, even though translations clearly form an Euclidean space since $\mathbf{t} \in \mathbb{R}^3$, the $SE(3)$ forms a non-Euclidean manifold, because its rotational component, the $SO(3)$, is not an Euclidean space.

This group is often represented using homogeneous transformation matrices of the following form:

$$\mathbf{X} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \text{ where } \mathbf{X} \text{ is a } 4 \times 4 \text{ matrix.} \tag{7}$$

Thus, the $SE(3)$ can be classified as a subgroup of $GL(n+1)$.

2.3.4 Lie Groups

According to [15], it can be shown that in addition to obeying the group axioms, the $GL(n)$ and, thus, the $SO(n)$ and $SE(n) \subset GL(n+1)$ are smooth manifolds also. For this reason they are called *Lie Groups*. Lie Groups have the special property that we can define a coordinate chart $\varphi_p(x)$ for any $p \in M$ as follows:

$$\varphi_x(y) = \varphi_{id}(x^{-1}y), \tag{8}$$

where φ_{id} is the coordinate chart centered at the identity $\mathbf{id} \in M$.
Using $\varphi_p(x) = \varphi_x(x)$ and plugging Eq. 8 into Eq. 2, we find the following simplified expression for the manifold operator $\boxplus$:

$$\boxplus : M \times \mathbb{R}^n \to M, \tag{9}$$
$$x \boxplus \delta = x^{-1}\varphi_{id}^{-1}(\delta). \tag{10}$$

Similarly, the simplified operator for $\boxminus$ in Eq. 4 becomes:

$$\boxminus : M \times M \to \mathbb{R}^n, \tag{11}$$
$$y \boxminus x = \varphi_{id}(x^{-1}y). \tag{12}$$

2.4 Representations of Poses

Whenever a rigid body transformation is used to encode the position and orientation of some object, it is referred to as the *pose* of this particular object. We have seen in the previous section that we might represent these members of $SE(3)$ as 4×4 matrices, but in practise this representation needs 16 floating-point values to be stored in a computer. It would be desirable to represent members of $SE(3)$ with less memory and still harness the power of the operations provided by matrix algebra. We cannot do much regarding the translation part, so it stays as a translation vector $\mathbf{t} \in \mathbb{R}^3$. Next, we need to find a good way of representing members of $SO(3)$, i.e. rotations. We could for example use the minimal representation for rotations, which consists of three Euler angles, e.g. roll, pitch and yaw, each one corresponding to a rotation in each three-dimensional axis. However, this representation is known to be problematic as it is prone to the gimbal lock problem, and also we are not able to compose several rotations very easily. An approach widely adopted in engineering is to use *rotation quaternions*. Although this is not a minimal representation, since quaternions live in $q \in \mathbb{R}^4$, an over-parametrised representation, they are a very elegant tool for representing rotations. They do not suffer from gimbal lock problem, and we can benefit from quaternion algebra when composing rotations without having to convert them into matrix form. A rotation of θ around an axis $\mathbf{u} \in \mathbb{R}^3$ can be written as a unit quaternion as follows:

$$\mathbf{q} = e^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}. \tag{13}$$

If $\mathbf{q_u}$ is a rotation of 30 degrees around the axis $\mathbf{u}$ (clockwise when we look at the same direction as $\mathbf{u}$), then the composite $\mathbf{q_u q_u}$ is a rotation of 60 degrees around the same axis. Also, the inverse rotation $\mathbf{q_u}^{-1}$ is given by the conjugate quaternion $\mathbf{q}^{\star}_{\mathbf{u}}$.

The rotation matrix corresponding to a unit quaternion $\mathbf{q}$ is given by

$$\mathbf{R_q} = \begin{bmatrix} 1-2q_j^2-2q_k^2 & 2(q_iq_j - q_kq_r) & 2(q_iq_k + q_jq_r) \\ 2(q_iq_j + q_kq_r) & 1-2q_i^2-2q_k^2 & 2(q_jq_k - q_iq_r) \\ 2(q_iq_k - q_jq_r) & 2(q_jq_k + q_iq_r) & 1-2q_i^2-2q_j^2 \end{bmatrix}. \tag{14}$$

Therefore, a pose can be represented by a vector $\mathbf{x} \in \mathbb{R}^7$, with three coordinates corresponding to the translation $\mathbf{t}$ and four corresponding to the unit quaternion $\mathbf{q}$, such that

$$\mathbf{x} = (t_x, t_y, t_z, q_x, q_y, q_z, q_w)^{\mathrm{T}}, \tag{15}$$

with identity element

$$\mathbf{id} = (0, 0, 0, 1, 0, 0, 0)^{\mathrm{T}}. \tag{16}$$

Finally, observe that we can describe Eq. 15 in matrix form as

$$\mathbf{X} = \begin{bmatrix} \mathbf{R_q} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{17}$$

2.5 Operations on Poses

When working with poses or rigid body transformations, we can define some useful notation to refer to common operations we can perform on them. First we define the *motion composition operator* $\oplus$ as

$$\mathbf{x_i} \oplus \mathbf{x_j} = \begin{bmatrix} \mathbf{q_i}(\mathbf{t_j}) + \mathbf{t_i} \\ \mathbf{q_i q_j} \end{bmatrix}, \text{where } \mathbf{q_i}(\mathbf{t_j}) = \mathbf{R}_{\mathbf{q_i}}\mathbf{t_j}. \tag{18}$$

Lastly, we define the *differencing operator* $\ominus$ as

$$\mathbf{x_i} \ominus \mathbf{x_j} = \begin{bmatrix} \overset{\star}{\mathbf{q}}_\mathbf{j}(\mathbf{t_i} - \mathbf{t_j}) \\ \overset{\star}{\mathbf{q}}_\mathbf{j}\mathbf{q_i} \end{bmatrix}, \text{ where } \overset{\star}{\mathbf{q}}_\mathbf{j}(\mathbf{t_i} - \mathbf{t_j}) = \mathbf{R}^{\mathrm{T}}_{\mathbf{q_j}}(\mathbf{t_i} - \mathbf{t_j}). \tag{19}$$

These operators are just shorthand definitions to the basic operations here denoted in matrix form:

$$\mathbf{X_i} \oplus \mathbf{X_j} = \mathbf{X_i}\mathbf{X_j}, \tag{20}$$

$$\mathbf{X_i} \ominus \mathbf{X_j} = \mathbf{X_j}^{-1}\mathbf{X_j}. \tag{21}$$

The advantage of using Eq. 18 and 19 is that all operations happen in the over-parametrised space $\mathbf{x} \in \mathbb{R}^7$ representation defined by us using quaternions. This facilitates the notation and the probabilistic formulation of the SLAM problem using the ML method on manifolds, as we shall see later.

2.6 Maximum Likelihood Method

In the ML method, given a measurement z, we are interested in finding x such that the posterior probability $p(x|z)$ is maximum assuming no prior knowledge about the behaviour of x. In other words, we want to find x that best explains our input measurement z as below:

$$\overset{*}{x} = \arg\max_x p(x|z). \tag{22}$$

Using the Bayes rule we can rewrite this statement as

$$p(x|z) = \eta p(z|x)p(x), \tag{23}$$

$$p(x|z) \propto p(z|x)p(x), \tag{24}$$

where η is a normalizing constant, $p(z|x)$ is our measurement model that tells how likely it is to make a measurement z given x, and $p(x)$ is the probability distribution that represents our prior knowledge about x. Since η is a positive constant that will not influence the maximization, we can rewrite Eq. 23 as

Eq. 24. And finally, if we assume we have no prior knowledge about the probability distribution of x, i.e. $p(x)$ is a uniform distribution, we can simplify Eq. 24 and write it is as

$$p(x|z) \propto p(z|x). \tag{25}$$

Therefore, under the assumptions of the ML method, we just showed that in order to solve Eq. 22 we will actually need to solve

$$\overset{*}{x} = \arg\max_{x} p(z|x). \tag{26}$$

We can also assume that z is normally distributed, which means that it represents a function of x with some added Gaussian noise, as given by Eq. 27:

$$z = f(x) + \omega, \text{ where } \omega \backsim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}^{-1}). \tag{27}$$

We can rewrite Eq. 27, and define

$$e_z(x) = z - \hat{z} = \omega, \tag{28}$$

where $\hat{z} = f(x)$, represents an expected measurement of x.

According to Eq. 27 and Eq. 28, $e_z(x) \backsim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}^{-1})$. However, this is only true if $e_z(x)$ is a linear function, which in practise is very commonly not the case. The reason why $e_z(x)$ has to be linear for remaining Gaussian distributed is because the distortions introduced by non-linearities in $e_z(x)$ destroy the nice bell shape property of the Gaussian distribution. A more complete demonstration of this fact is given by [23] when deriving the equations for the Kalman filter, which makes exactly the same assumption. In order to overcome this issue we linearise $e_z(x)$ via a first order Taylor expansion around $x = x_0$. Note that this approximation is exact when $e_z(x)$ is already linear, thus we are actually generalizing our formulation to accept non-linearities in $e_z(x)$. The linearization is given by

$$e_z(x_0 + h) \cong e_z(x_0) + \mathbf{J}h, \text{ with } \mathbf{J} = \left.\frac{\partial e_z(x)}{\partial x}\right|_{x=x_0}. \tag{29}$$

Since now $e_z(x)$ is linearised, then $e_z(x) \backsim \mathcal{N}(\mathbf{0}, \mathbf{\Omega}^{-1})$. We can therefore write $p(z|x)$ as

$$p(z|x) = g(e_z(x_0) + \mathbf{J}h, \mathbf{0}, \mathbf{\Omega}^{-1}), \tag{30}$$

where $g(x, \mu, \mathbf{\Sigma})$ is a multivariate Gaussian with mean μ and covariance $\mathbf{\Sigma}$, and

$$g(x, \mu, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right). \tag{31}$$

Now, because we approximated $e_z(x)$ by its first order Taylor expansion, we are not able to calculate x that maximises g directly, but we can find the increment h that maximises g. In order to do that, we can take the log-likelihood of g:

$$\mathcal{L}(x, \mu, \mathbf{\Sigma}) = \ln g(x, \mu, \mathbf{\Sigma}) = -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \mathbf{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}). \tag{32}$$

As shown in Eq. 30, setting $x = e_z(x_0) + \mathbf{J}h$, $\mu = \mathbf{0}$ and $\mathbf{\Sigma} = \Omega^{-1}$, where Ω is the information matrix of the system, we arrive at Eq. 33. The reason for making the information matrix explicit will become clear in the next chapter when we define the Graph SLAM problem. Note that we maximise g by minimizing its log-likelihood $\mathcal{L}$. Finally, note that the problem has now become the least-square minimization of $\mathbf{x}^{\mathrm{T}}\mathbf{\Omega}\mathbf{x}$, essentially proving that finding x with maximum probability $p(x|z)$ in an ML sense is the same as minimizing the quadratic error of $\mathbf{x}^{\mathrm{T}}\mathbf{\Omega}\mathbf{x}$, thus showing that these two problems are actually the same.

$$\mathcal{L}(x, \mu, \mathbf{\Omega}^{-1}) = -\frac{1}{2}(e_z(x_0) + \mathbf{J}h)^{\mathrm{T}} \mathbf{\Omega} (e_z(x_0) + \mathbf{J}h). \tag{33}$$

Thus, in order to minimise $\mathcal{L}$ we can take its derivative with respect to h and set it to $\mathbf{0}$:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{h}} = \mathbf{J}^{\mathrm{T}} \mathbf{\Omega} e_z(x_0) + \mathbf{J}^{\mathrm{T}} \mathbf{\Omega} \mathbf{J} h = \mathbf{0}, \tag{34}$$

$$\mathbf{J}^{\mathrm{T}} \mathbf{\Omega} \mathbf{J} h = -\mathbf{J}^{\mathrm{T}} \mathbf{\Omega} e_z(x_0). \tag{35}$$

Hence, in order to find the increment h that maximizes g, we need to solve the linear system of the form $\mathbf{A}x = b$ shown in Eq. 35, where $\mathbf{A}$ is a positive definite symmetric matrix. This allows us to find fast solutions by using Cholesky decomposition, for example.

Once the increment h is found using Eq. 35, the current estimate x_0 is updated by $x_1 = x_0 + h$, and the process is repeated until h is sufficiently small.

2.6.1 Maximum Likelihood for Multiple Independent Measurements

It is often the case we have multiple independent measurements at once instead of just one. In this case we are trying to estimate the posterior as below:

$$p(x|z_1,\ldots,z_n) = p(x|z_{1:n}) = \eta p(z_1,\ldots,z_n|x)p(x), \tag{36}$$

$$p(x|z_{1:n}) \propto p(z_1,\ldots,z_n|x)p(x), \tag{37}$$

$$p(x|z_{1:n}) \propto p(z_1,\ldots,z_n|x), \tag{38}$$

$$p(x|z_{1:n}) \propto \prod_{i=1}^{n} p(z_i|x). \tag{39}$$

Similarly as before, we start from the Bayes rule in Eq. 36, then in Eq. 37 we can drop the constant η since it is not going to influence the maximization, later in Eq. 38 we drop the term $p(x)$, assuming uniform prior for x. Finally, we can arrive at Eq. 39 under the assumption that all measurements $z_1,\ldots,z_n$ are conditionally independent.

If we take the log-likelihood of $p(x|z_{1:n})$ and follow the same steps as before, we arrive at the following simplified expression:

$$(\sum_{i=1}^{n} \mathbf{J_i}^{\mathrm{T}} \mathbf{\Omega_i} \mathbf{J_i}) h = \sum_{i=1}^{n} \mathbf{J_i}^{\mathrm{T}} \mathbf{\Omega_i} e_{z_i}(x_0). \tag{40}$$

This simplified Eq. 40 is again a linear system of the form $\mathbf{A}x = b$ and $\mathbf{A}$ is a symmetric matrix. Once more, this system can be solved very quickly using Cholesky decomposition.

3 Related Work

Perhaps a lunatic was simply a minority of one.

George Orwell, 1984

In the literature, a large variety of solutions to the SLAM problem were proposed. These approaches can be classified into two main categories, filtering and smoothing approaches [10]. In this chapter we will briefly discuss these approaches in order to provide a short overview about the state of the art.

3.1 Filtering Approaches

Filtering approaches deal with SLAM as a state estimation problem in which the state vector includes the pose of the robot and all landmark positions, which are used to represent the map. The state vector is incrementally updated as the robot gathers measurements from the physical world, and these measurements are integrated into the current state of the robot, yielding a better estimation of the map and robot's pose in such map. Approaches like these are, hence, incremental in nature and also commonly referred to as online SLAM methods. Examples of such methods are particle filters, Kalman and information filters. In fact, all these filtering methods are different instantiations of the Bayesian filter [23], which is a mathematical framework that allows us to model our belief about the world and improve this belief incrementally via measurements performed on the physical world.

Approaches based on filtering were the first practical solutions to the SLAM problem [17]. EKF SLAM was a popular method among these approaches and was typically based on landmarks. This approach used an extended Kalman filter to estimate the posterior probability distribution of the robot pose together with all landmark positions. Such idea was first proposed and implemented by [21]. More recently, MonoSLAM became popular by applying the same principles using an RGB monocular camera [6]. However, EKF-based approaches suffer from serious scalability issues. The reason is that every time a landmark is re-observed all other landmarks need to be updated as well, which leads to a computational cost of $O(N^2)$ for updating the state vector, where N is the number of landmarks. This quadratic complexity limits the practical use of such approaches to maps containing only a few hundred landmarks, although in real scenarios this number reaches a few millions.

FastSLAM proposed a solution to this scalability drawback by decomposing the problem into a robot pose estimation problem and a collection of

landmark position estimation problems that are conditioned on the estimate of the robot's pose [16]. The key idea was the realisation that landmark measurements are actually conditionally independent given a robot trajectory. Thus, by using a particle filter for estimating the trajectory of the robot and individual Kalman filters for landmark position estimates, this method could overcome many issues presented by EKF-based approaches. This is because landmark updates do not depend on each other anymore, yielding a final complexity of $O(M \log K)$ using a tree data structure for optimisation, where M is the number of particles in the filter and K the number of landmarks.

FastSLAM represented a remarkable improvement over EKF-based approaches. Further optimisations were made by [11], who used a grid map representation and reduced the number of particles required for a proper functioning of the approach. Nonetheless, although FastSLAM is theorically capable of operating in 3D space, the number of particles required to cover a good proportion of the space becomes prohibitive in practise. Because of this, EKF-based approaches are usually the easiest choice, despite their limitation, for achieving practical results, such as the aforementioned work [6].

3.2 Smoothing Approaches

In contrast to filtering approaches, smoothing approaches optimise the whole trajectory of the robot based on the full set of measurements gathered throughout its history. Because of this, these approaches are often said to address the full SLAM problem, typically using least-square error minimization techniques [10]. The great advantage of these approaches is that the optimisation can be revised and rectified, since the full set of data is kept throughout the whole mapping process. In addition, the complexity of these approaches grows only linearly on the size N of the trajectory, i.e. $O(N)$. Smoothing approaches commonly represent a robot's trajectory as a graph, where each node is a pose, and each edge represents a spatial constraint between these nodes. The task of these approaches is to construct, manage and finally find an optimised graph that is the most likely trajectory made by the robot given the set of measurements, overcoming the cumulative odometry error.

Throughout the last decade a number of successful Graph SLAM approaches have been implemented following this line of thinking. Many approaches focused on devising efficient implementations of back-end optimisation techniques, assuming a consistent graph was already constructed by a front-end, a module responsible for dealing with all incoming sensory data. Some researchers worked towards the formalisation of the problem and effi-

cient implementation of solutions considering the theory of smooth manifolds and the sparse structure of the problem. This was the case with the Sparse Least Squares on Manifolds (SLoM) back-end [12] and the General (Hyper) Graph Optimization, so called g2o framework [13]. Other researchers focused on improving the ability of correcting very large trajectories using stochastic gradient descent [18] or extending this idea using a tree parametrisation to improve convergence speed, such as the TORO back-end [11].

A few approaches focused on the construction of a front-end using RGBD sensors. Examples of such approaches are the RGB-D SLAM [7] and RTAB-Map [14] systems. The former was based on the g2o back-end, used a Dijkstra projection approach for computing loop closures, and Iterative Closest Point (ICP) as odometry source, whereas the latter allowed the usage of either TORO or g2o as back-end, used a Bayesian classifier for finding loop closures, and had multiple sources of odometry, including feature-based odometry using Bag-of-Features (BOW) and ICP. Another Graph SLAM approach that used g2o as a back-end for graph optimisation was the Large Scale Direct Monocular SLAM (LSD-SLAM), which used a visual odometry algorithm based on a photo-consistency energy function [8] and FAB-MAP as loop closure detector [5].

The FAB-MAP approach is in itself a SLAM approach that operates not in the metric space, but in the appearance (visual) space [5]. This approach is widely used for place recognition and topological mapping. And thus, it is a very useful tool for finding loop closures. In this work we use FAB-MAP for exactly this purpose.

4 Graph SLAM System

I always do that, get into something and see how far I can go.

Richard Feynman

An intuitive way of thinking about SLAM is using a graph data structure, so called *pose graph* [10]. In this graph, vertices are poses where the robot has been and edges connecting these poses represent relative transformations between them, along with a measure of how certain we are regarding this relative transform. Every edge in this graph represents a constraint for the optimisation method, providing evidence regarding the true trajectory of the robot. An illustration of this pose-graph can be seen in Fig. 1.

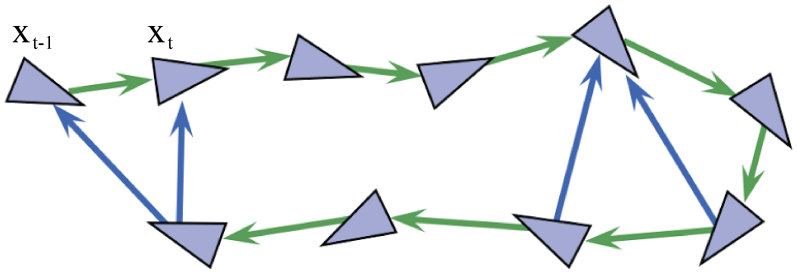

Figure 1: Pose graph representation of SLAM. Green links between nodes x_t and x_{t-1} are odometry edges or constraints. Blue links between nodes x_i, x_j represent occasions where nodes in the graph are revisited or seen again from another position. These edges are called loop closure constraints.

A Graph SLAM algorithm can be divided into two sub-modules responsible for performing two main tasks. The first module is the back-end. This module is responsible for receiving this abstract representation of the problem in the form of a *pose graph* and finding the trajectory or graph that best explains the measurements represented as constraints in the graph. This optimisation procedure is formalised in a least-square sense, derived from a ML formulation. The second module is the front-end. The front-end of a Graph SLAM algorithm is responsible for dealing with the graph construction, identification of constraints and its maintenance.

Note that the back-end is completely agnostic to the type of sensors being used, the hardware and many other details regarding the graph construction and management. In contrast, the front-end design is very sensor dependant, since we need to know what kind of sensory data is going to be utilised in order to devise approaches to transform this incoming data into the graph

that is later passed to the back-end. Furthermore, it is important to construct a consistent graph, since the least-square optimisation of the back-end is very sensible to outliers, e.g. false loop closure links, or wrong constraints in general. An abstract diagram of the system is depicted by Fig. 2.

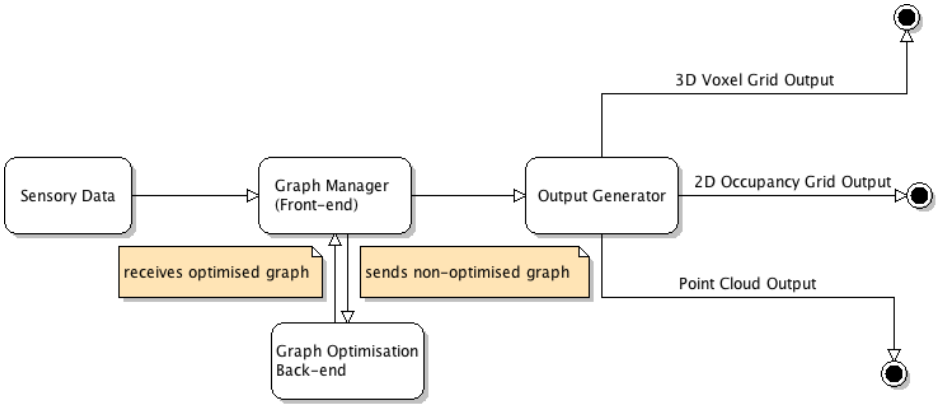

Figure 2: Abstract diagram of a Graph SLAM system.

In the remaining of this chapter we will describe how these two modules are defined and how they work together in our Graph SLAM system.

4.1 Back-end

In this section we will define how the back-end finds the optimal trajectory in a ML sense. In summary, let $\mathcal{C}$ be the set of pairs (i, j) for which a measurement $\mathbf{z}_{ij}$ exists, then $\mathbf{z} = \{\mathbf{z}_{ij} | (i, j) \in \mathcal{C}\}$, i.e. $\mathbf{z} = [\mathbf{z}_{i_1 j_1}, \mathbf{z}_{i_2 j_2}, \dots]$. We want to find $\mathbf{x}$ such that $p(\mathbf{x}|\mathbf{z})$ is maximum. The vector $\mathbf{x}$ is the concatenation of all poses $\mathbf{x}_i$, representing the trajectory of the robot such that

$$\mathbf{x} = (\mathbf{x_1}^{\mathrm{T}}, \mathbf{x_2}^{\mathrm{T}}, \dots)^{\mathrm{T}}. \tag{41}$$

Therefore, we want to find

$$\overset{*}{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{x}|\mathbf{z}_{i_1 j_1}, \mathbf{z}_{i_2 j_2}, \dots), \tag{42}$$

which, as we have seen in Section 2.6, under the set of ML assumptions, is equivalent to

$$\overset{*}{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{z}_{i_1 j_1}, \mathbf{z}_{i_2 j_2}, \dots | \mathbf{x}). \tag{43}$$

4.1.1 Maximum Likelihood Formulation

The central notion of "additive" Gaussian noise in Eq. 28 that enabled us to derive the equations for the ML method in Section 2.6 does not make sense anymore, since the + operation is not well defined on manifolds. The ML method must take into account the fact that we are dealing with members of $SE(3)$ which, as we have seen, form a manifold. Therefore, since we chose to represent poses in an over-parametrised space $\mathbf{x} \in \mathbb{R}^7$, and we also defined the operators $\boxplus, \boxminus, \oplus$ and $\ominus$, we can define our error function as

$$e_{z_{ij}}(x_i, x_j) = e_{ij}(x) = z_{ij} \boxminus \hat{z}_{ij} = z_{ij} \boxminus (x_j \ominus x_i). \tag{44}$$

Fig. 3 depicts the error function of Eq. 44 in addition to the other quantities involved in the definition of a graph constraint.

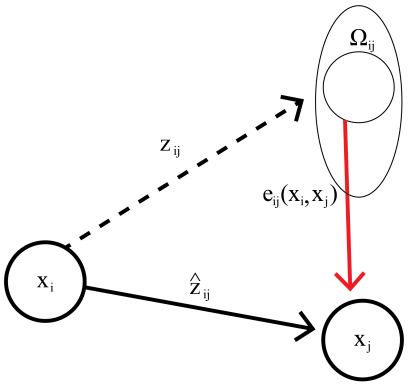

Figure 3: Graph SLAM representation.

Our Taylor expansion from Eq. 44 also needs to be updated using the $\boxplus$ operator to:

$$e_{ij}(x_0 \boxplus h) \cong e_{ij}(x_0 \boxplus \mathbf{0}) + \mathbf{J_{ij}} h = e_{ij}(x_0) + \mathbf{J_{ij}} h. \tag{45}$$

Note that

$$\mathbf{J_{ij}} = \frac{\partial e_{ij}(x \boxplus h)}{\partial h} = \frac{\partial e_{ij}(x)}{\partial x} \cdot \frac{\partial x \boxplus h}{\partial h}\bigg|_{h=0}. \tag{46}$$

Observe that we can obtain the manifold Jacobian of Eq. 46 above simply multiplying Eq. 29 by the partial derivative $\frac{\partial x \boxplus h}{\partial h}\big|_{h=0}$ (using the chain rule for partial derivatives).

Therefore, we can show analogously to Section 2.6 that we will need to solve

$$(\sum_{(i,j)\in\mathcal{C}} \mathbf{J_{ij}}^{\mathrm{T}}\mathbf{\Omega_{ij}}\mathbf{J_{ij}})h = \sum_{(i,j)\in\mathcal{C}} \mathbf{J_{ij}}^{\mathrm{T}}\mathbf{\Omega_{ij}}e_{ij}(x_0), \tag{47}$$

where $\mathcal{C}$ is the set of constraints found by the front-end (edges between nodes x_i, x_j), and the Jacobian $\mathbf{J_{ij}}$ is given by Eq. 46.

At last, we are able to solve the linear system of Eq. 47, which has the usual form $\mathbf{A}x = b$. Solving it yields the desired increment $\mathbf{h}$ which is used to update our current estimate, keeping in mind that we are dealing with a manifold now. Thus, $x_1 = x_0 \boxplus h$. This process is repeated until h is sufficiently small.

Some final practical considerations regarding memory allocation and the sparse structure of the matrices involved in Eq. 47 have to be taken into account to design an efficient implementation of this optimisation procedure. For this reason we have decided to use the g2o open-source framework that implements exactly this optimisation method. The framework is called g2o, which stands for General (Hyper) Graph Optimization [13]. Since this framework has been widely used by the scientific community [7, 14, 8], reimplementing this procedure seemed pointless to our studies. Furthermore, the decision of using this framework instead of implementing our own enabled us to focus on practical issues regarding the front-end implementation.

4.2 Front-end

The front-end is equally, or perhaps even more important than the back-end for a Graph SLAM algorithm. Due to the back-end least-square nature, even just a single outlier constraint introduced in the graph might compromise the whole optimisation result. This is because outliers will represent large squared errors according to the ML formulation. Therefore, it is the role of the front-end to manage the sensory data, and ensure the consistency of the generated graph, which will be passed to the back-end.

4.2.1 General Principle

The front-end has the following main tasks to accomplish:

Add odometry constraints Whenever sufficient movement is detected between two consecutive odometry readings an odometry constraint is

added to the underlying graph along with a new vertex. The information acquired at this particular position (e.g. depth and color images) is also kept and described for later data association purposes.

Detect loop closures Using the data collected throughout its trajectory, the front-end needs to be able to tell if a particular place was seen before and which previous place it was.

Add loop closure constraints Given some candidate loop closure detection between nodes x_i and x_j, the front-end has to find a relative transformation between x_i and x_j using the data collected at these specific spots.

Evaluate and monitor the graph Because wrong loop closure constraints (outliers) can be mistakenly added to the graph, it is required to have a procedure to periodic check the consistency of the added loop closures. Invalid loop closures must be deleted or disabled. This is a step suggested by [18, 17]. However, due to time constraints this procedure was not implemented in this work.

4.2.2 System Design

The Graph SLAM system was implemented using the g2o framework for graph optimisation [13], and FAB-MAP [5] for loop closure detection. The final system was deployed and tested on the Robot Operating System [1] infrastructure. A class diagram of the system is depicted in Fig. 4.

As shown in Fig. 4, the `SLAM` class comprises the whole Graph SLAM procedure. Whenever some new data arrive (color and depth images, odometry and laser readings), this data are incorporated in the `DataPool` as a `DataSpot` object and connected with the previously added `DataSpot` via an odometry link. A `DataSpot` object might have links with others besides odometry links, that is the reason why a `DataPool` object has a relationship with a `FABMAP` object, which detects loop closures using visual features present in the color image of a given `DataSpot`. Whenever a loop closure is found, the `TransformEstimator` uses the class `DataSpotMatcher` for matching two `DataSpots` that were said to form a loop by `FABMAP`, and estimates the relative transform between them. To estimate this relative transformation, the class `TransformEstimator` uses the corresponding 3D points of the matched 2D feature points, and tries to find a transformation $\mathbf{T} \in SE(3)$ that best aligns these two sets of corresponding points using Random Sampling Consensus (RANSAC) [9]. The constructed Graph is sent to the `GraphOptimiser` back-end periodically for optimisation.

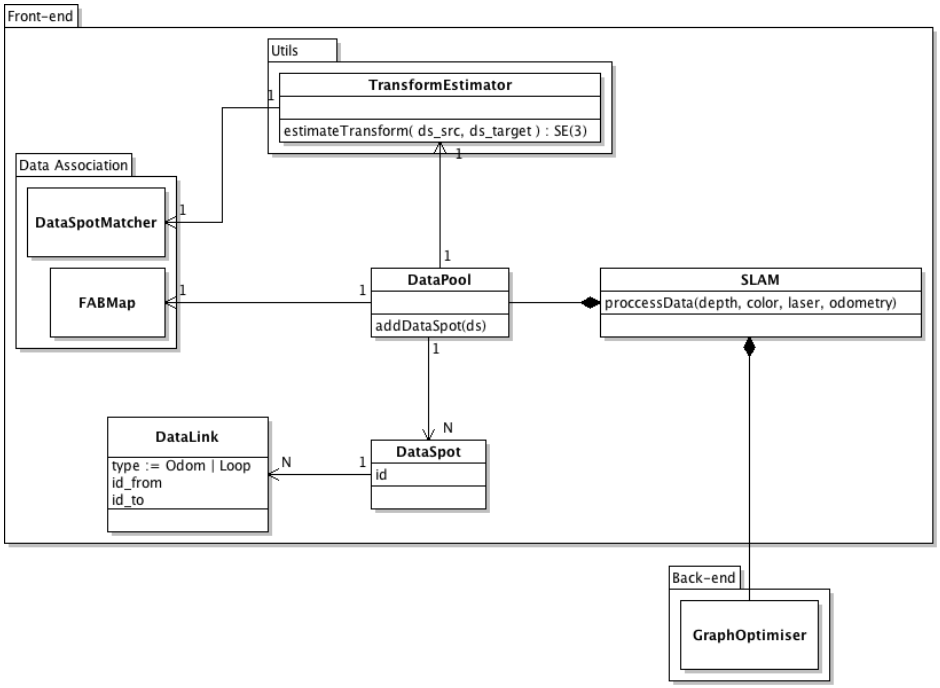

Figure 4: Simplified class diagram of the system.

4.2.3 Odometry Constraints

The odometry source for the current version of the system comes from the wheel encoders of the mobile platform utilised (described in Chapter 5). Each time a new data arrives, the odometry is checked to see if there was enough movement from the robot. We considered that if the robot moved 1 centimetre or rotated at least 0.005 radians, then it moved enough. If this is the case, then a new node is added to the underlying graph together with an odometry link. Thus, if the robot previously was at $\mathbf{x_{t-1}}$ and now the encoder reports the odometry pose $\mathbf{x_t}$, then pose $\mathbf{x_t}$ will be added as a new node, along with the data collected at that spot. In addition, the odometry link $(\hat{\mathbf{z}}_{\mathbf{t-1t}}, \mathbf{\Omega}_{t-1t})$ is be added, where $\hat{\mathbf{z}}_{\mathbf{t-1t}} = \mathbf{x_t} \ominus \mathbf{x_{t-1}}$ and $\mathbf{\Omega}_{t-1t}$ is computed as in Section 4.2.4 description of the information matrix computation.

4.2.4 Loop Closure Constraints

The process for adding loop closure constraints can be summarised in the following steps:

Loop closure detection Whenever a new node $\mathbf{x_i}$ is added to the graph, a loop closure detection routine based on FAB-MAP is run. Whenever

this routine detects a loop closure, then the identifier of node $\mathbf{x_j}$ is returned.

Relative transform estimation Having identified a loop between nodes $\mathbf{x_i}$ and $\mathbf{x_j}$, the relative transform $\mathbf{z}_{ij}$ between these two nodes is estimated. In order to estimate $\mathbf{z}_{ij}$ we first extract a set of visual features $\mathbf{f}^i = \{f^i{}_1, f^i{}_2, \dots\}$ and $\mathbf{f}^j = \{f^j{}_1, f^j{}_2, \dots\}$ from the color images associated with $\mathbf{x_i}$ and $\mathbf{x_j}$, respectively. We refer to a node $\mathbf{x_k}$ with its associated data as a `DataSpot`, in the context of the system design. Let $S = \{(f^i{}_{k_1}, f^j{}_{k_1}), (f^i{}_{k_2}, f^j{}_{k_2}), \dots\}$ be the set of matching features between the two nodes (i, j), and let $kp^i = kp^i(f^i), kp^j = kp^j(f^j) \in \mathbb{R}^2$ be the key-point positions in the images of the nodes (i, j). Because we also have access to the depth images of nodes (i, j), we can map all key-points to their corresponding 3D points $p^i = p^i(kp^i), p^j = p^j(kp^j) \in \mathbb{R}^3$ in space, yielding $P = \{(p^i{}_{k_1}, p^j{}_{k_1}), (p^i{}_{k_2}, p^j{}_{k_2}), \dots\}$. Finally, a point-to-point model registration based on RANSAC method is used to estimate the best alignment $\mathbf{z}_{ij}$ between the matching points in P, i.e. $\mathbf{z}_{ij} = \arg\min_{\mathbf{z}} \sum_{m=1}^{n} \|p^i_{k_m} - \mathbf{z}p^j_{k_m}\|$.

Loop closing If the procedure above converges, then we compute the variance of the alignment, which consists of ${\sigma_{ij}}^2 = \frac{\sum_{m=1}^{n} \|p^i{}_{k_m} - \mathbf{z}_{ij} p^j{}_{k_m}\|^2}{n-1}$. Then, setting the information matrix to $\mathbf{\Omega}_{ij} = I \frac{1}{{\sigma_{ij}}^2}$, the resulting loop closure link will be $(\mathbf{z}_{ij}, \mathbf{\Omega}_{ij})$.

5 Experiments

> If it can be destroyed by the truth, it deserves to be destroyed by the truth.
>
> Carl Sagan

In order to proof test the implemented approach, a sequence of experiments was performed. The first experiment used a well-known benchmark for RGBD SLAM evaluation [22]. The second one was performed in an indoor domestic environment. Both experiments and results are described in the remaining of this chapter. First we will describe the hardware utilised to test the algorithm, then we will continue to describe the benchmark experiment, followed by the domestic environment experiment.

5.1 System Hardware

The hardware used to run the implemented system is as follows:

Robot The system was tested on a domestic vacuum cleaner robot, a Neato XV-14 Signature made by Neato Robotics. A picture of this robot can be seen in Fig. 5. It proved to be a very cheap and programming-friendly mobile platform. The robot is equipped with a low-cost LIDAR and wheel encoders. An open-source driver available for the Robot Operating System [1] interfaced the robot hardware providing odometry and laser scan readings.

RGBD sensor An ASUS XTion PRO LIVE depth sensor (Fig. 6) was mounted on the robot, which provided registered depth images and color images as input to the system.

Computer The computer hardware utilised consisted of a laptop with an Intel Core i7 @ 1.6 GHz processor, 6 GB RAM and equipped with an NVIDIA GeForce GT 330M graphics card.

A picture of the robot setup can be seen in Fig. 7.

Figure 5: Neato XV-14 robot used.

Figure 6: ASUS Xtion PRO LIVE RGBD sensor used.

Figure 7: Robot setup with ASUS Xtion PRO LIVE mounted on top of Neato XV-14.

5.2 Benchmark Experiment

This benchmark experiment used the publicly available dataset and evaluation tools described in [22]. We particularly made use of the robot SLAM sequences, making comparisons with the provided ground-truth trajectory.

5.2.1 Freiburg Dataset Experiment

In this experiment we used the robot SLAM sequence named "freiburg2 pioneer slam2". This sequence was recorded from a kinect mounted on a Pioneer robot, which was joysticked through a large hall. The ground-truth trajectory length is 21.735 meters. We compared the non-optimised trajectory error in Fig. 8 with respect to the ground-truth. Then, we compared the optimised trajectory shown in Fig. 9 with the ground-truth. Finally, we made a two-tailed *t-test* in order to verify if the improvement using the implemented approach was statistically significant. When we refer to non-optimised trajectory we mean that the trajectory was generated using solely the odometry of the robot.

5.2.2 Results

The final experiment's result showed that, although the odometry seemed to be, unexpectedly, rather precise already, adding our Graph SLAM approach slightly improved the final trajectory error, making it drop a few centimetres. Nonetheless this result also showed that our procedure for loop closing could not handle very well the low number of features provided by the benchmark dataset: only 4 far-in-time loop closures were found, from a total of 277 loop closures. Since nodes are labelled with increasing integer identifiers, we defined loops connecting two nodes x_i, x_j to be far loops if $j - i >= 10$, otherwise they are considered near-in-time loop constraints. This limited number of far-in-time loop closures suggested us to look into more robust loop closing strategies as future work for this kind of environment. Overall, the approach seemed to work as theoretically expected given the set of assumptions made during its design, in particular the choice of using a hybrid approach with a visual-based detection for loop closing and relative transformation computation. Whenever few far-in-time loop closures are detected, the system degrades expectedly as if it was using only odometry, although the overall error is smoothed locally due to the close-in-time (local) loop closure constraints.

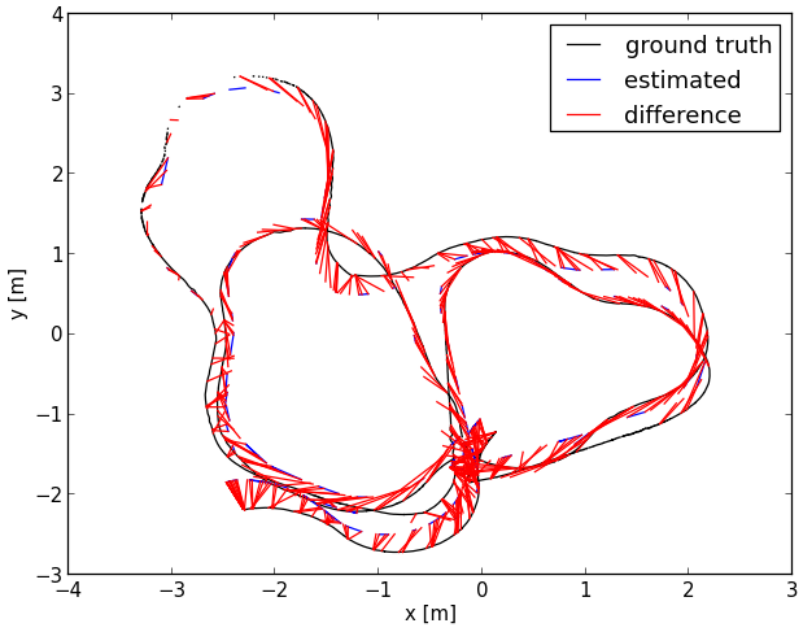

Figure 8: Non-optimised trajectory statistics: trajectory mean error 0.335751 m, RMSE 0.370089 m, std 0.155682 m, number of pairs 452.

5.3 Domestic Environment Experiment

In this section we perform a few experiments on indoor domestic environments. We then make a qualitative analysis of the final mapping result and robot's trajectory.

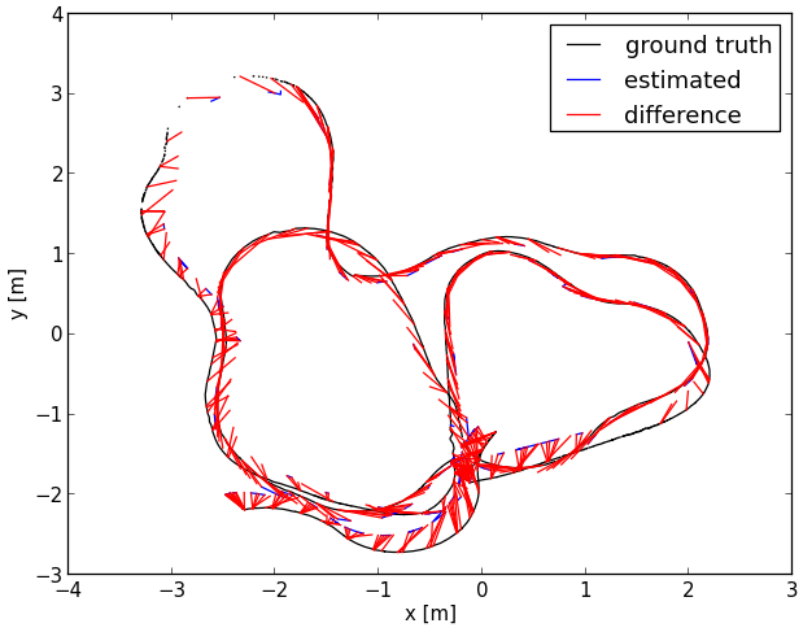

Figure 9: Optimised trajectory statistics: trajectory mean error 0.305305 m, RMSE 0.330452 m, std 0.126441 m, number of pairs 462. Performed t-test showed a two-tailed P value equals to 0.0012 with respect to Fig. 8. This difference is considered to be very statistically significant with 95% confidence. The total number of far-in-time loop closures was 4 and the total number of loop closures was 277.

5.3.1 Mapping a Flat

For this experiment we manually navigated the robot through a flat in the order specified in Fig. 10. The whole trajectory size was approximately 40 meters long. We often turned the robot 360 degrees in order to capture as much discerning features of the environment as possible, in order to maximise the chances of finding loop closures as we revisit places of the flat. The total number of 573 loop closures were found, of those 474 were near-in-time loop closure constraints (local loop closures), and 99 were far-in-time loop closures (global loop closures). In the non-optimised trajectory shown in Fig. 11, the loop closures are labelled in gold color. Far-in-time loop closures are more evident than near-in-time loops, which can still be spotted by the cyan squares marking loop closure detections. This distinction was made only for the purpose of having an idea of how important far-in-time loop closures might possibly be, since the back-end does not make any such distinctions

Figure 10: Flat visiting order overlay on top of optimised occupancy grid map. The flat was visited following the order 1-2-3-4-5-6, and then traced back its steps going through 6-5-4-3-2-1.

between constraints. It is expected that farther in time loop closures will reduce the overall trajectory error more noticeably due to the cumulative nature of the odometry error.

The data collected manoeuvring the robot around the flat were recorded and played twice. The first time had the graph optimisation turned off, which means the final map shown in Fig. 11 was made only using robot's odometry. The second time the recorded data were played with the graph optimisation turned on and the optimised trajectory can be seen in Fig. 12.

5.3.2 Results

The final result was quite positive. We can see clearly the cumulative odometry error being shown in Fig. 11, whereas Fig. 12 shows a fairly more consistent map and trajectory. Another aspect that could be observed during the execution of the experiments was that farther in time loop closures were responsible for large trajectory corrections, which is in accordance to the mathematical formulation of the back-end and empirical observed behaviour of odometry error.

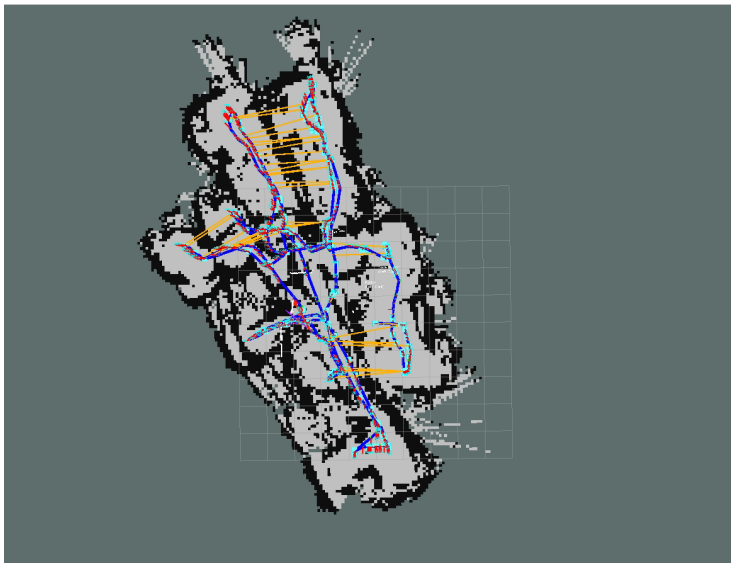

Figure 11: Non-optimised trajectory and occupancy grid map using odometry only. Robot trajectory is highlighted in dark blue, loop closure constraints are golden and loop detection points are cyan.

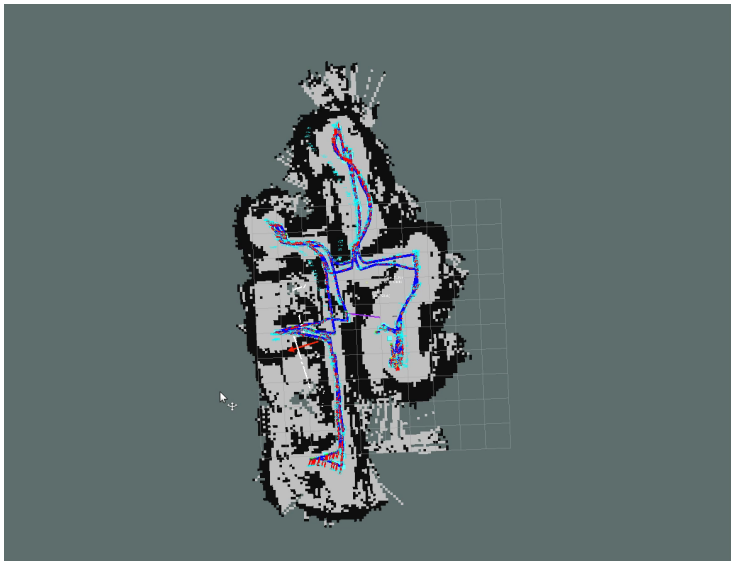

Figure 12: Optimised trajectory and occupancy grid map using all constraints.

6 Conclusion

> But I think of what I have done and thought during the day. Thus ruminating, patient as a cow, I ask myself: What were thy ten overcomings? And what were the ten reconciliations, and the ten truths, and the ten laughters with which my heart enjoyed itself?
>
> Friedrich Nietzsche, Thus Spoke Zarathustra

In this work we implemented a complete Graph SLAM system able to construct consistent maps of the environment, as demonstrated by the performed experiments. More than that, we were able to organise a solid foundation with which we can move towards in several research directions. The implemented system also needs improvements, particularly regarding the creation of a more robust loop closure constraint estimation procedure, which proved to have difficulties in certain environments. The implemented system showed satisfactory, results working successfully in indoor environments.

6.1 Future Work

The system still has room for improvements. In this final Section we list a few points that we wish to work in the future.

Improve loop closure computation We noticed that the implemented system showed low number of far-in-time loop closure constraints. Due to the good performance the system showed in the indoor domestic environment, we concluded that this was due to our visual dependant approach for loop closure constraint estimation. As a future work, it is a top priority improvement to be done in the system to improve the robustness of our approach towards featureless environments.

Add other odometry sources Additional sources of odometry will enable us to test the system this time in a full 3D trajectory. For this reason we intend to develop visual feature-based procedures, such as BOW odometry employed by [14] and depth-based approaches such as an ICP-based odometry employed by [7].

Implement efficient memory management Another potential improvement is to add the ability of dealing with multi-session mapping and very long trajectories. In order to cope with this, we ponder that we will need to add a more sophisticated memory management method for the `DataPool`. A human inspired mechanism used by [14] based on the works of [3] and [4] is an example of work in this direction.

References

[1] The robot operating system (ros). `http://www.ros.org`. Accessed: July 2015.

[2] Toyota tour guide robot. `http://www.toyota.co.jp/en/news/07/0822.html`. Accessed: July 2015.

[3] R. C. Atkinson and R. M. Shiffrin. Human memory: A proposed system and its control processes. *In K. W. Spence and J. T. Spence (Eds.), The Psychology of learning and motivation: Advances in research and theory (vol. 2)*, pages 89–105, 1968.

[4] A. D. Baddeley. *Human Memory: Theory and Practice*. Lawrence Erlbaum Associates, Hove, UK, 1990.

[5] M. Cummins and P. Newman. Highly scalable appearance-only SLAM - FAB-MAP 2.0. In *Proceedings of Robotics: Science and Systems*, 2009.

[6] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[7] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on*, 30(1):177–187, Feb 2014.

[8] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. *In European Conference on Computer Vision (ECCV)*, 2014.

[9] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[10] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *Intelligent Transportation Systems Magazine, IEEE, vol. 2, no. 4, pp. 31–43*, 2010.

[11] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, 2007.

[12] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds. Diploma thesis, University of Bremen, 2008.

[13] R Kummerle, G Grisetti, H Strasdat, K Konolige, and W Burgard. G2o: A general framework for graph optimization. In *Proceedings Robotics and Automation (ICRA)*, 2011.

[14] M. Labbé and F. Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. *In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[15] J. M. Lee. *Introduction to Smooth Manifolds*. Springer, 2002.

[16] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*. AAAI, 2002.

[17] F. Neuhaus. A full 2d/3d graphslam system for globally consistent mapping based on manifolds. Diploma thesis, Universität Koblenz-Landau, 2011.

[18] E. Olson. Recognizing places using spectrally clustered local matches. *Robot. Auton. Syst.*, 57(12):1157–1172, December 2009.

[19] E. Prassler and K. Kosuge. Domestic robotics. *Springer Handbook of Robotics, pp. 1253-1281*, 2008.

[20] W. R. Scott, G. Roth, and J. Rivestçois. View planning for automated three-dimensional object reconstruction and inspection. *ACM Comput. Surv*, 35(1):64–96, March 2003.

[21] R. C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainly. *Int. J. Rob. Res.*, 5(4):56–68, December 1986.

[22] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[23] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[24] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohb, Cedric Dupont, Lars erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe Van Niekerk, Eric Jensen, Gary Brad-ski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: the robot that won the darpa grand challenge. *Journal of Field Robotics, 23:661-692*, 2006.

[25] H. Liu W. Tan, Z. Dong, G. Zhang, and H. Bao. Robust monocular slam in dynamic environments. *In Mixed and Augmented Reality (ISMAR), IEEE International Symposium on, pp. 209–218*, 2013.

Signatures

Supervisor: Veronica Teichrieb	Date
Student: Ermano Ardiles Arruda	Date