Preface

This Bachelor assignment is written in the context of the organization Revolve NTNU, during the spring 2018, as a part of their Driverless team. The intention of the assignment is to document the work we have done and provide a reflected view on what can be improved, with the choice of our approach to SLAM in mind. This assignment is also written to provide a tangible and intuitive description of SLAM and iSam2, for easy reference for future Revolvers.
The algorithms used to produce the results in this assignment were implemented in collaboration with Lars Gustavsen. The implementation of iSam2 deploys the software developed by Georgia Tech., called GTSAM.

Table of Contents

List of Figures

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) is concerned with the problem of navigating a mobile robot in an unknown environment while simultaneously creating a map of its environment, [4]. Solving this problem includes detecting landmarks, performing data associations, mapping of landmarks and localizing the robot. The combination of these into a single approach that manages to *incrementally* update the state of a robot and its environment, has proved essential for deploying autonomous cars, unmanned aerial vehicles (UAVs), etc, into the world, see e.g. [18], [15], [9]. Implementing a solution to the SLAM problem requires taking into account the probabilistic nature of sensor noise, thus rendering statistics a necessary tool. Examples are methods using Rao-Blackwellised particle filtering [11], Extended Kalman Filtering [5] or Sparse Extended Information Filters [17].

The reason why statistics is important within SLAM can be illustrated by a brief example of a car equipped with a camera and GPS, see fig. (1.1). The camera provides local observations of surrounding objects whereas the GPS provides measurements of the position and orientation of the car in a global frame. These observations arrive independently at different rates, each with its own amount of clutter and noise. The measurements of *where* objects are located relative to the car along with the GPS measurements themselves are therefore inherently uncertain.

To account for the uncertainty there are usually two approaches that come to mind;

1. Assume that the local environment surrounding the car is known so the task is rather to adjust the location of the car in that environment, provided local measurements from the camera and the GPS. This approach is known as *localization*.

2. Assume that position and attitude of the car are perfectly known so it becomes possible to create a map of the environment based on local measurements of surrounding objects. This approach is known as *mapping*.

The problem with the former is that a predetermined map may be unattainable, outdated or simply erroneous. Assuming a known environment will in these cases not help localize

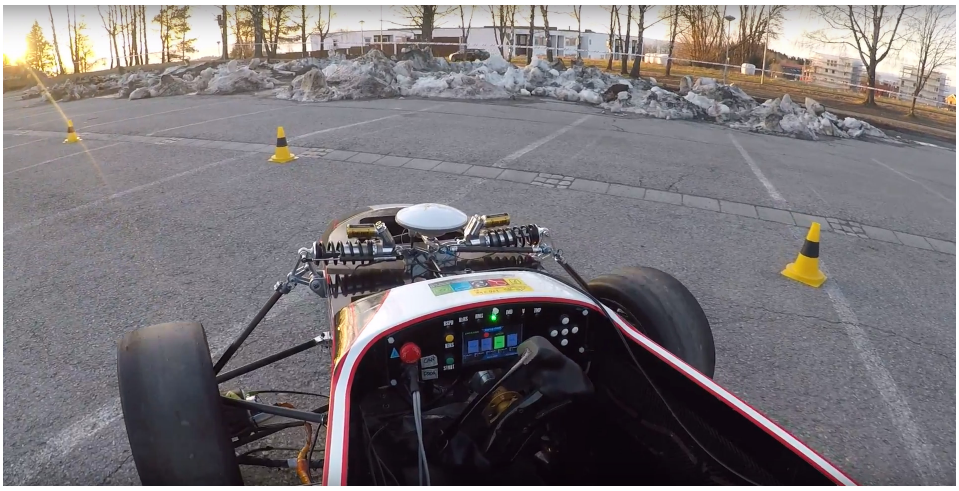


Figure 1.1: The view from one of the cameras on Eld (Revolves autonomous racing car), where the cones represent what will later be referred to as landmarks. The white disc on the front dampers are the GPS receiver.

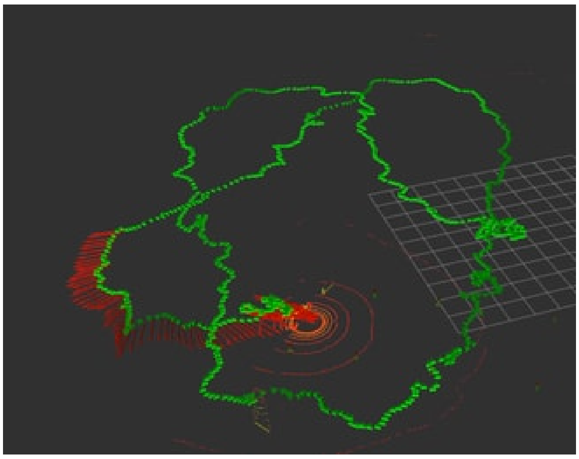


Figure 1.2: Visualization of how odometry drifts for a car driving in a large oval. The odometry is provided using lidar with the method in [21].

the car correctly relative to local measurements. The latter problem depends purely on the assumption that the car has perfect knowledge of its position and orientation. Consider a case where the GPS is affected by drift and noise, but the car is still designed to naively create a map of its environment. Now the perceived path taken by the car will not be able to bite its tail due to drift, see fig. (1.2) for an illustration. The map will therefore not line up correctly and result in an inconsistent estimate.

This Bachelor assignment will describe the Statistical structure underlying SLAM based on [4], and the state-of-the-art solution, incremental Smoothing and Mapping 2 (iSam2), see e.g. [7]. ISam2 is used by Revolve NTNU, a non-profit student organization centered around building Formula Student racing cars, as part of the design of our perceptual system for the new driverless car. The algorithm provides a way to incrementally improve how the car maps its environment which provides a perfect fit for real-time operation. It is therefore natural to understand the benefits and drawbacks of the algorithm, as well as the potential modifications and additional assumptions that can be made in order to optimize it for our purpose.

The assignment will be structured as follows. Chapter 2 will define notation and basic concepts within statistics that is necessary in order to understand SLAM and iSam2. The introduction to the various topics will be brief assuming basic knowledge from probability theory and statistics. Chapter 3 will give a more thorough description of SLAM. chapter 4 will present iSam2 in order to obtain a clear picture of how it works. Chapter 5 will discuss the possible modifications that can be done to a naive SLAM approach using iSam2 that is beneficial in our application of racing around an unknown track. Results from real-time testing will be provided in this chapter along with the improvements that could be done. Chapter 6 will discuss the results along with further work.

Chapter 2

Theoretical background

This chapter provides a quick reference for some of the tools that will be used in this assignment. Sec. 2.1 contains basic statistical notation that will be used. The purpose is solely to avoid misinterpretations and to ease the process of understanding the material. Sec. 2.2 provides a quick tour through the assumptions and properties for graphical models that are essential for the understanding of iSam2.

2.1 Notation

- A *set* of outcomes is denoted without bold font according to $\theta_{k:k+n} := (\theta_k, \theta_{k+1}, .., \theta_{k+n})$.
- Bold font, $\boldsymbol{\theta}_{k:k+n} := [\theta_k, \theta_{k+1}, .., \theta_{k+n}]^T$, denotes a vector of variables defined over the span of several discrete time instants, here from k to $k+n$.
- A multivariate normal distribution is represented by $N(\boldsymbol{\theta}_{k:k+n}; \boldsymbol{\mu}, \Sigma)$, where $\boldsymbol{\theta}_{k:k+n}$ is a vector of outcomes $\theta_{k+i} \in R^n$ from a normal distribution with mean $\boldsymbol{\mu}$ and covariance Σ.
- A joint probability distribution function (joint pdf) will be represented by $p(\theta_{1:2})$ and the conditional probability distribution function (conditional pdf) by $p(\theta_2|\theta_1)$.
- An outcome θ_k is represented by either a *robot pose* or a *landmark* in the context of SLAM. These outcomes are in most practical applications vectors and are therefore denoted without bold font.
- A robot pose is a vector describing position and orientation, for example $x_k = (x, y, \psi)^T$ for 2D or $x_k = (x, y, z, \phi, \omega, \psi)^T$ for 3D. The rotations ϕ, ω, ψ represent respectively roll, pitch and yaw.
- A landmark is a vector describing objects surrounding the robot using points in the same euclidean space as the robot $l_k = (x, y)^T$ or poses similar to that of the robot $l_k = (x, y, \psi)^T$, here in 2D. This depends solely on the object. A cone is a good

example of an object that can be represented by a point since its orientation doesn't matter. A human being walking past the robot can on the other hand be practical to represent using a pose to understand which way he is walking.

- The map m is a set gathering all known landmarks.
- Sensor measurements is denoted by z_i representing measurement i.
- The notation $z_{i,k}$ denoting measurement i that happened to occur at time step k, is used if time dependence has to be deployed.

As mentioned, notation is simplified by using normal-font letters for single robot poses x_k, landmarks l_k, measurements z_i and for the map m.

2.2 Graphical models

Graphical models are useful in the sense that they are able to provide intuition about large complex systems and that they can display dependencies between variables easily overlooked. To some extent they can display data in an intuitive way trying to resemble the way of human reasoning. The Bayesian network (Bayes net), can be considered as such a model as discussed extensively in [12]. Another approach is that of Markov networks (Markov nets) which is discussed extensively in [12] and [14], taking into account the numerical aspects and their relation to the information matrix. The presentation of the graphical models in this chapter are meant to provide the necessary foundation required to understand the Bayes tree.

2.2.1 Factor graphs

A factor graph represents the joint pdf $p(\theta_{1:n})$ using a bipartite graph. Two kind of nodes are represented in the graph; variable nodes represented by the variables $\theta_{1:n}$ and factor nodes represented by the functions $f_1(\Theta_1), f_2(\Theta_2), .., f_m(\Theta_m)$, where $f_i : \Theta_i \rightarrow R$. The sets $\Theta_i \subset \Theta$ consists of variable nodes θ_k adjacent to the factor node f_i. The factor graph describes a factorization of the joint pdf

$$p(\theta_{1:n}) = \prod_i f(\Theta_i) \tag{2.1}$$

Using a factorization of the joint pdf to represent the state of the world provides the opportunity to update it in an incremental fashion.

A practical example

It provides an accurate visual representation of the interdependence between variables. An example is how a robot translates forward in space. Consider the car from the introductory example moving in a 2D plane with a known initial pose $x_0 = (x, y, \psi)^T = (2, 2, 0)^T$ and a measured velocity of 1 m/s in the x-direction. This initial pose can look like x_0 in fig. (2.1). The function factor on the left hand side of x_0 specifies the knowledge of this

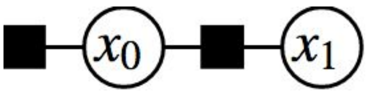


Figure 2.1: Small factor graph example expressing an initial pose x_0 with a prior factor node and a consecutive pose x_1 connected to x_0 with a *between* factor. The initial factor can be called a unary factor since it is connected to one pose only.

initial information. Given that there are 1 s between consecutive velocity observations, it is possible to deduce that the car will move one meter, with uncertainty, in the x-direction before the velocity is measured again at $x_1 \approx (3, 2, 0)$. Thus, the factor graph will display how these two variables, the poses x_0 and x_1 in fig. (2.1), are connected by a function factor. This results in the factorization

$$\begin{aligned} p(\theta_{1:2}) &= p(x_0, x_1) \\ &= f(x_0) f(x_0, x_1) \end{aligned} \tag{2.2}$$

Information propagation

In practice it is desirable not only to represent the consecutive poses of the car with a factorized joint probability distribution, eq. (2.1), but to actually infer the value of specific poses. In the context of factor graphs this is done through information propagation, see .e.g [10], which can be understood as a way to pass information between nodes in a graph.

2.2.2 Chordal Bayesian networks

A Bayes net is a directed acyclic graph (DAG), $G \in (\Theta, E)$, where Θ is the set of vertices and E is the set of directed edges connecting these vertices. This graph provides an unambiguous representation of dependency between variable sets. The graph is decoded as a factorization of the joint pdf

$$p(\theta_{1:n}) = \sum_{i \in \Theta} p(\theta_i | \theta_{\text{pa}(i)}) \tag{2.3}$$

where $\theta_{\text{pa}(i)}$ denotes the parent node of θ_i. The Bayes net in fig. (2.2) will for example describe the joint pdf of $x_0, x_1, x_2, x_3, l_0, l_1, l_2$ in factorized form

$$p(x_{1:2}, l_{1:2}) = p(x_0|x_1)p(l_0|x_1)p(l_1|x_{1:2})p(x_1|x_2)p(x_2|x_3)p(l_2|x_3)p(x_3) \tag{2.4}$$

The power of such a representation is the ability to display conditional independencies. Knowing the variable x_2 will render the variables x_1 and l_1 conditionally independent from x_3. Formally, this is defined as the *d-separation* criterion, [12], which permits us to determine by inspection how two variable sets are independent from each other given a third set.

A *chordal Bayes net* is a graph where every cycle of size four or more has a *chord*. This chordal pattern results in a graph consisting of triangles, between x_1, x_2 and l_1, and

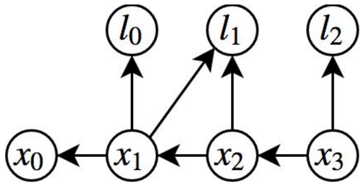


Figure 2.2: Bayes Net representing causal dependencies between variable nodes in *reverse order*.

straight connections, between x_1 and x_2, fig. (2.2). These two sets of connections will in succeeding Chapters be referred to as *cliques*. Also notice that the arrows are pointing from vertices with a high count to those with lower count, which can informally be described as a *reverse order* representation. This representation occur because of the way the Bayes net will be constructed in subsequent chapters.

2.2.3 Gaussian Markov fields and fill-in

A Gaussian Markov Random Field (GMRF) is defined by a random vector $\mathbf{\Theta}_{k:k+n}$ on a labelled graph $G = (X, E)$ with mean μ and precision matrix $Q > 0$, iff its density has the form

$$\pi(\boldsymbol{\theta_{k:k+n}}) = (2\pi)^{-n/2}|Q|^{1/2} \exp\Big(-\frac{1}{2}(\boldsymbol{\theta}_{k:k+n} - \mu)^T Q(\boldsymbol{\theta_{k:k+n}} - \mu)\Big) \tag{2.5}$$

and

$$Q_{ij} \neq 0 \quad \iff \quad \{i, j\} \in E \quad \forall i \neq j \tag{2.6}$$

[14]. The precision matrix is defined as $Q = \Sigma^{-1}$ and will therefore be equivalent to the so called information matrix in the SLAM literature. The importance of GMRF in the context of this assignment lies within the interpretation of the underlying graph G as a direct representation of the information matrix Q. Only an essential part of the couplings between the underlying graph in a GMRF and the information matrix will be elaborated in this section, since only a subset of the class of graphs is used.

The underlying graph is an *undirected graph* representing conditional independence between variables, i.e. a Markov net. In the information matrix Q, conditional *dependence* is decoded as non-zero elements, see fig. (2.3). The relation between a Markov net and the *square root information matrix*, formally defined as the triangular matrix R in $Q = R^T R$, is defined in [14] to not be clear-cut. The triangular matrix R is either equally or more dense than the information matrix due to the appearance of *fill-in*. A fill-in is a non-zero element appearing in R that is non-zero in Q. Fig. (2.3) displays several important facts that are important to notice. Firstly, how a fill-in term appears in the square root information matrix is shown by the read block in the lower triangular matrix for the non-chordal Markov net. This is due to the conditional dependency structure in the graph and the variable ordering describing how variables are stored in the information matrix [14]. A broader discussion of these effects on the square root information matrix fill-in will be delayed until ch. (4).

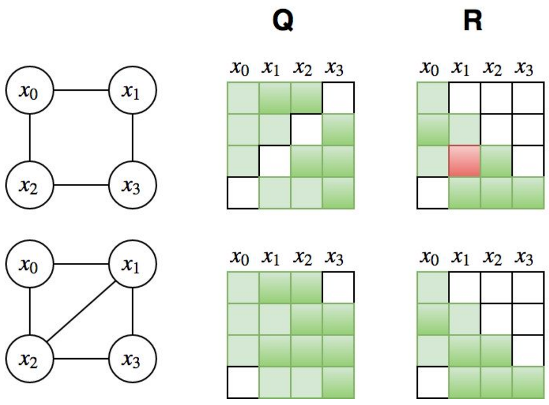


Figure 2.3: A visual representation of the difference between fill-in in a non-chordal and chordal Markov Net, with its corresponding information matrix and square root information matrix respectively. The colored squares are non-zero elements whereas the white squares are zero elements. The red square is a non-zero fill-in.

Furthermore, the difference between non-chordal and chordal Markov nets and their relationships to their respective square root information matrices are also shown in fig. (2.3). The chordal Markov net has the property that it can directly display the appearing fill-in provided a *reasonable* variable ordering. Since conditional dependencies displayed by chordal directed and undirected graphs are equal [12], it is important to keep in mind for subsequent chapters that the square root information can have a direct graphical representation.

Chapter 3

The structure in SLAM

There is extensive literature on the different approaches to solving the SLAM problem which should be consulted in order to gain a complete overview, see e.g. [2], [4], [16]. This chapter will focus on presenting the core concepts defining the underlying structure in SLAM rather then the different methodologies for solving it.

First, a conceptual explanation about SLAM will be presented to make the rigorous statistical description more digestible. Then, the statistical description of the structure in SLAM will be presented. Lastly, how to obtain an estimated solution using maximum a posteriori (MAP) estimation will be explained using the previously defined concept of a factor graph, see sec. (2.2.1).

3.1 Introducing SLAM

There are general assumptions and concepts in the complete description of SLAM that are important to understand before diving into the more theoretical content. These assumptions and concepts will be presented next.

3.1.1 The motion model

The motion model

$$p(x_k|x_{k-1}, u_k) \tag{3.1}$$

predicts how the robot pose will evolve between the discrete time instants $k-1$ to k given a control input u_k, i.e. it is a state transition model. A practical implementation would be a function of odometry and/or velocity, usually a simplified model, with the goal of providing a crude estimate. A simple example is illustrated by the discrete time state transition model

$$x_k = x_{k-1} + u_k + w_k \tag{3.2}$$

where $w_k \sim N(w_k; 0, \Sigma_k)$ is zero mean Gaussian white noise with a diagonal covariance matrix Σ_k. The covariance matrix defines the uncertainty in the model and has to specified by an engineer in practice.

Errors are introduced in this prediction step due to both model assumptions and drift caused by measurement errors. Drifting can be defined as an error that gradually builds up over the lifetime of the robot.

3.1.2 The measurement model

The measurement model

$$p(z_k|x_k, m) \tag{3.3}$$

describes the probability of performing a certain measurement z_k given the current robot pose x_k. For simplicity it is usually modelled as normal distribution. Note that the measurement model has to be modified to fit the existing sensors on the robot. Therefore errors are introduced due to faults in sensor measurements and due to inaccuracies in estimating the current pose of the robot.

How these models are explicitly utilized in SLAM will be presented later. For now it suffices to understand that there are uncertain aspects regarding the above models as well as the sensor measurements themselves.

3.1.3 Prior assumptions

To build up a map while simultaneously localizing where a mobile robot is, requires a couple initial definitions

- The robot has to start out with a *known* position and orientation, such that it can be modeled as prior information about the robot pose, x_0.
- The map has to be built in a predefined *coordinate frame*.

The coordinate frame is defined with respect to the application at hand. If the goal is to navigate using a global navigation satellite system (GNSS) like GPS, it is desirable to use a earth fixed coordinate frame that is consistent with the GPS framework. The realization of such an approach implies that the prior pose of the robot can be represented by the earth fixed coordinates of its starting point. The motion model, eq. (3.1), is then used to describe how the robot manoeuvres relative to the earth. On the other hand, it may be desirable to avoid using a GPS system due to either price or its unreliability. Either way, a more *local* coordinate frame can be used to gradually build up the map. In this case it is trivial to define the prior location of the car since the placement of the coordinate frame relative to the car is left to the engineer. For the sake of illustration it will from here on be assumed that a GPS isn't available, thus the second approach is deployed with an assumed initial pose at the origin.

3.1.4 The necessity of static landmarks

Having a known starting point it is now possible to manoeuvre around in the environment with the intention of estimating the current pose as accurately as possible. Naively this can be done using the motion model, eq. (3.1), complemented with velocity measurements. For example, the current pose can be predicted by integrating the translational and rotational velocities using the motion model, and adding them to the previous pose estimate. The issue with this approach is that the simplistic nature of the state transition model is unaccounted for. The estimated error will therefore drift as shown in fig. (1.2). Since a perfect model is, in general, unattainable, a natural extension is to exploit information from the existing scene, portrayed by exteroceptive sensors, to adjust the robot pose. The techniques for doing so vary between sensors, but results, if done correctly, in increased positional accuracy, see e.g. [21] for a more detailed description using lidar measurements. The upside of exploiting this extra source of information is the possibility of correcting some of the drift and uncertainty that arise in the initial scenario. The downside is that there is still drift and uncertainty, implying that the end result will end up like before, fig. (1.2).

To remove this gradually increasing drift, a reliable source of information has to be available that can tell the robot exactly where it is. For example, consider yourself navigating towards the Eiffel Tower in the city Paris. The enormous landmark is always in the corner of your eye until you decide to walk past some large city blocks. You navigate through these blocks for some time taking a couple of right and left turns until you forget the exact direction you were headed in the first place. You feel disoriented and lost before the Eiffel Tower suddenly appear at some distance ahead of you. Just as for human beings, robots can correct the estimate of their pose if they have a relation to the position of *static landmarks*. The question that arise is then; How can the robot exploit existing landmark positions to correct its own pose, when they are a priori unknown?

3.1.5 SLAM and loop closures

As the robot moves ahead to explore new areas it will observe landmarks at a consistent rate. These landmarks are incrementally updated in the current map estimate relative to the estimated trajectory of the robot. Since the robot pose gradually drifts, the landmarks have correspondingly increasing positional uncertainty the later they are observed. Re-observed landmarks can therefore be used to adjust previous estimates computed since the last time the landmark was observed. This concept of propagating information backwards to correct erroneous estimates is in the literature defined as performing a *loop closure*.

By re-exploring previously visited areas the map gradually converge toward a ground truth solution under the assumption that landmarks are static. The treatment of both non-static landmarks and wrongly added landmarks, called outliers, are important when designing a SLAM algorithm. These nuances are unfortunately not trivial to handle and not crucial for understanding SLAM. They are therefore only mentioned as important factors to be aware of in this assignment.

3.2 The structure of SLAM

From secs. (3.1.4) and (3.1.5) there appear to be an incremental structure to the update of both the robot pose estimate and the map estimate. Furthermore, sec. (3.1.5) describe how both these estimates are corrected at certain points in time. This section will describe how these corrections are performed and provide the statistical framework to describe the incremental update procedure.

3.2.1 The basics

In all its glory, SLAM can be reduced to identifying the posterior pdf

$$p(x_{0:k}, m | z_{0:n,0:k}, u_{0:k}, x_0) \tag{3.4}$$

where k denotes the current discrete time step and x_0 is prior information on the robot pose, see sec. (3.1.3). The posterior pdf in eq. (3.4) is defined as the estimate of the map and the trajectory of the robot pose from the beginning of time. Trying to estimate this distribution at every time step will both be infeasible in real-time and unnecessary. It is important to understand that only a subset of the robot poses and landmarks are affected in every time step (this is most likely even in the presence of loop closures). To obtain a more feasible incremental approach, the problem are reduced to computing the joint posterior pdf

$$p(x_k, m | z_{0:n,0:k}, u_{0:k}, x_0) \tag{3.5}$$

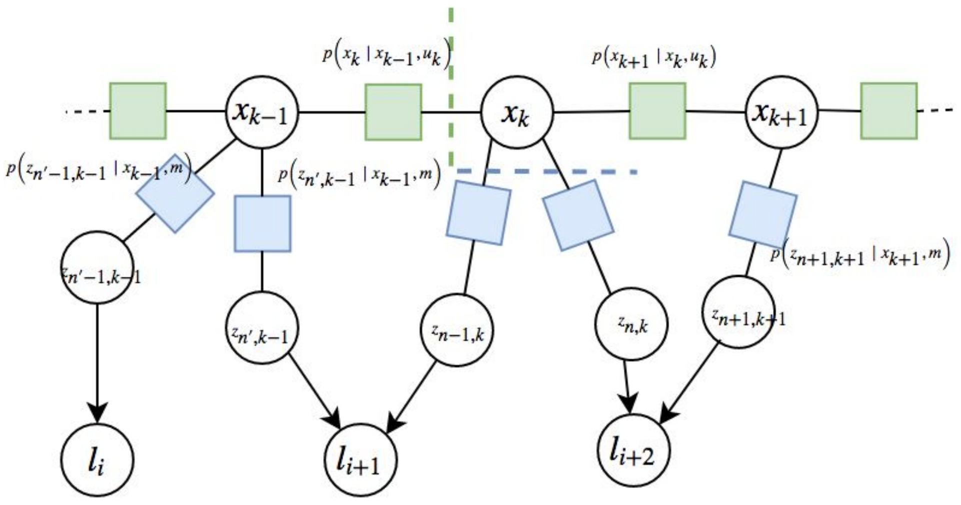
at all points k in time. This solution requires knowledge from the previous step $p(x_{k-1}, m | z_{0:n',1:k-1}, u_{0:k-1}, x_0)$, where $0 < n' < n$. Estimating the robot pose and landmark positions at time k then boils down to identifying eq. (3.5) using a two-step recursive procedure. This procedure consists of a time-update step that provides a prediction of the robot pose and the updated landmark positions.

$$p(x_k, m | z_{0:n',0:k-1}, u_{0:k}, x_0) = \int p(x_k | x_{k-1}, u_k) p(x_{k-1}, m | z_{0:n',0:k-1}, u_{0:k-1}, x_0) dx_{k-1} \tag{3.6}$$

which can be thought of as propagating all previous information forward in time. In fig. (3.1), this is seen as propagating all information known before x_k past the green dotted line at time k. The second part of the procedure is the measurement update that corrects the predictions made in eq. (3.6) based on incoming measurements

$$p(x_k, m | z_{0:n,1:k}, u_{0:k}, x_0) = \frac{p(z_{n':n,k} | x_k, m) p(x_k, m | z_{0:n',0:k-1}, u_{0:k}, x_0)}{p(z_{n':n,k} | z_{0:n',0:k-1}, u_{0:k})} \tag{3.7}$$

This correction step exploits new information from the sensors about existing landmarks, sec. (3.1.5), to correct errors propagated through previous predictions. In fig. (3.1) this can be seen as using the new information provided below the dotted blue line along with existing map information.



$p(x_k \mid x_{k-1}, u_k)$
$p(x_{k+1} \mid x_k, u_k)$
x_{k-1}
x_k
x_{k+1}
$p(z_{n'-1,k-1} \mid x_{k-1}, m)$
$p(z_{n',k-1} \mid x_{k-1}, m)$
$p(z_{n+1,k+1} \mid x_{k+1}, m)$
$z_{n'-1,k-1}$
$z_{n',k-1}$
$z_{n-1,k}$
$z_{n,k}$
$z_{n+1,k+1}$
l_i
l_{i+1}
l_{i+2}

Figure 3.1: Hidden Markov model illustrating the differentiation between time-updates, propagating past the green lines, and measurement updates, exploiting new information below the blue line. This figure is intentionally made to resemble a factor graph, thus the green boxes decode a motion model, eq. (3.1) and the blue boxes decode a measurement model, eq. (3.3).

Before proceeding it can be enlightening to receive a more rigorous explanation of the difference between the concepts of localization and mapping than in the introductory chapter. The concepts are defined separately in [4] to describe why they cannot be considered as disjoint problems in a SLAM solution. The same will be done in this assignment beginning with the localization problem

$$p(x_k | z_{0:n,0:k}, u_{0:k}, x_0, m) \tag{3.8}$$

It can be seen that the localization problem makes explicit us of the map m, eq. (3.8), thereby assuming that landmark positions are known with certainty. The goal is therefore to compute the robot pose with respect to these landmarks. Secondly the mapping problem

$$p(m | x_{0:k}, z_{0:n,0:k}, u_{0:k}, x_0) \tag{3.9}$$

makes explicit use of the exact knowledge of the robot poses to create the map. The answer to the question regarding why localization and mapping cannot, in general, be partitioned according to

$$p(x_k, m | z_{0:n,0:k}, u_{0:k}, x_0) \neq p(x_k | z_{0:n,0:k}, u_{0:k}, x_0) p(m | z_{0:n,0:k}, u_{0:k}, x_0) \tag{3.10}$$

becomes evident from their definitions, eqs. (3.8) and (3.9). Furthermore, it can be seen from the partition, eq. (3.10), that robot poses and landmarks are assumed *conditionally independent* provided measurements and controls. Due to the explicit dependencies, it is necessary to exploit a more coupled dependency structure between the robot poses and landmarks than the one above, to obtain *consistent* estimates. A prominent observation regarding such dependencies are about those between landmarks.

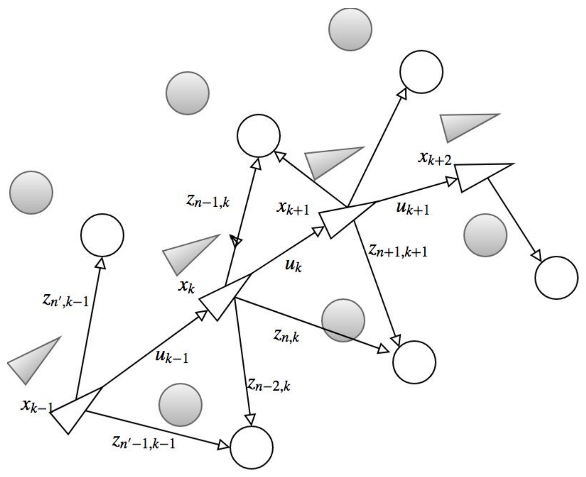


Figure 3.2: Visualization of absolute drifting error and how the relative landmark positions stay *nearly* unaffected by drifting in the estimated robot position. The colored circles and triangles are estimated and the white are ground truth. The triangles are the robot pose and the circles are landmarks.

3.2.2 Landmark correlations

A vital effect on the error introduced into the estimation of the incremental posterior pdf, eq. (3.5), is how the motion model and the measurement model are the cause of drift in landmark positions. Drift is introduced in the landmark position mainly due to uncertainty in the robot location. Spatially close landmarks are usually observed from closely related poses, thus implying that differences in the imposed drifting error is small. These landmarks are therefore highly correlated and are represented by highly peaked joint distributions $p(l_i, l_j)$, and more dispersed marginal distributions, $p(l_i)$ and $p(l_j)$. The goal of any SLAM algorithm would then be to monotonically increase its certainty in the relative landmark positions, gradually peaking $p(l_i)$ and $p(l_j)$. Intuitively this can be thought of as every landmark and pose being connected in a network of springs. As correlations between landmarks and poses increases, the springs become gradually stiffer until a point where they can be considered rigid.

It is important to notice that measurements of a set of landmarks are *nearly independent* of the relative location between landmarks. That is, the pose of the robot does not impose any additional constraints on the landmarks due to its slowly drifting error. Fig. (3.2) pictures how this error builds up for the robot pose with the aforementioned effect of just shifting the whole set of landmarks.

3.2.3 Loop closures

A *loop closure* occurs when a robot returns to a previously explored area. The robot inhabits a sense of awareness, loosely equivalent to; *I have been here before, thus my current map and position should have closed a loop leading me to this place. For some reason it*

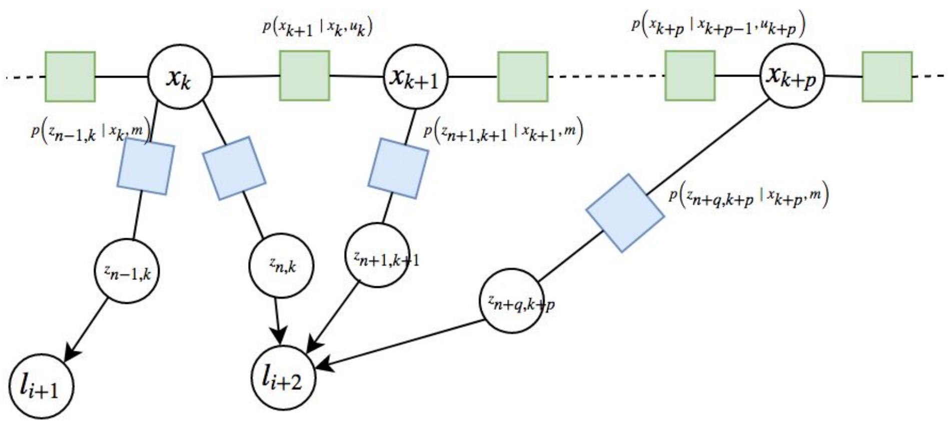

$p(x_{k+1} \mid x_k, u_k)$
$p(x_{k+p} \mid x_{k+p-1}, u_{k+p})$
x_k
x_{k+1}
x_{k+p}
$p(z_{n-1,k} \mid x_k, m)$
$p(z_{n+1,k+1} \mid x_{k+1}, m)$
$p(z_{n+q,k+p} \mid x_{k+p}, m)$
$z_{n-1,k}$
$z_{n,k}$
$z_{n+1,k+1}$
$z_{n+q,k+p}$
l_{i+1}
l_{i+2}

Figure 3.3: Hidden Markov model illustrating a loop closure after the pose has propagated p time steps forward in time. Since both pose x_k and x_{k+p} connects to l_{i+1}, all poses, and corresponding landmarks, estimated between them will be affected by the consecutive correction.

has not, so I should correct the errors that have propagated over time. Mathematically, loop closures shift the current pose to where it should be and propagates the information through the existing set of poses and landmarks, see fig. (3.3). Due to the property discussed in sec. (3.2.2), the correlation between landmarks are large, thus shifting a pose will *pull* all connected landmarks along with it. This is because the poses can be thought of as rigidly attached to their observed landmarks. In statistical terms, Cov(l_i, l_j) will stay consistently low while Var(l_i) turns out to increase for subsequent landmarks as the pose evolves. The correction that occur when a loop is closed results in a reduction in both for all landmarks affected.

3.3 MAP estimation

The SLAM problem was originally solved using nonlinear state estimation techniques, but results from recent years show that SLAM formulated in the context of MAP estimation give better results, [3]. Define the variable Θ including the sets of robot trajectories $x_{0:k}$ and landmarks m, i.e. the same as in the section on Factor graphs, sec. (2.2.1). The reason for this is because factor graphs provide a representation of the MAP estimation problem that allow understandable reasoning about interdependence among variables. Furthermore, the SLAM literature, see e.g. [3], use the notation z_i for measurements described by the function

$$z_i = h_i(\theta_i) + v_i \tag{3.11}$$

where $h_i(.)$ is a nonlinear function, in general, encapsulating both the motion model, eq. (3.1), and the measurement model, eq. (3.3). $v_i \sim N(v_i; 0, \Sigma_i)$ is zero mean Gaussian white noise with a diagonal covariance matrix Σ_i. The functions $h_i(.)$ are generally modelled as function factors in a factor graph, sec. (2.2.1).

Estimation of the optimal assignment Θ^* can now be formulated from a Bayesian perspective as

$$\begin{aligned}\Theta^* &= \underset{\Theta}{\operatorname{argmax}}\, p(\Theta|z_{0:n}, u_{0:k}, x_0) \\ &= \underset{\Theta}{\operatorname{argmax}}\, p(z_{0:n}|\Theta)p(\Theta)\end{aligned} \tag{3.12}$$

where the controls $u_{0:k}$ are discarded to simplify notation. Further, assuming conditionally independent measurements $z_{0:n}$ given Θ, i.e. uncorrelated noise, leads to

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, p(\Theta) \prod_i p(z_i|\Theta_i) \tag{3.13}$$

A representation resembling the factorization in a factor graph, eq. (2.1), has now been obtained considering that eq. (3.13) express a factorization of the joint pdf from eq. (3.12).

To obtain an explicit closed form expression for the above optimization problem further assumptions have to be made. The prior $P(\Theta)$ is assumed to have the form

$$p(\Theta) \propto \exp\Big(-\frac{1}{2}||h_0(\Theta) - z_0||^2_{\Sigma_0}\Big) \tag{3.14}$$

for some function $h_0(\cdot)$, prior mean z_0 and covariance matrix Σ_0. In the absence of prior information this can also be assumed to be uniform. The measurement likelihood is now assumed to be normally distributed with Σ_i being the covariance matrix for the current observation

$$p(z_i|\Theta_i) \propto \exp\Big(-\frac{1}{2}||h_i(\Theta_i) - z_i||^2_{\Sigma_i}\Big) \tag{3.15}$$

The measurement likelihood function are used for any measurement added to the graph. In this context, no explicit distinction is made between the motion model, eq. (3.1) and the observation model (3.3). The generality of representing both these models with the same measurement likelihood is a core strength of the factor graph abstraction.

Combining the aforementioned assumptions and applying the prior and measurement likelihoods, eqs. (3.14) and (3.15), to eq. (3.13) results in the expression

$$\begin{aligned}\Theta^* &= \underset{\Theta}{\operatorname{argmax}}\log\Big(p(\Theta)\prod_i p(z_i|\Theta_i)\Big) \\ &= \underset{\Theta}{\operatorname{argmin}} -\log\Big(p(\Theta)\prod_i p(z_i|\Theta_i)\Big) \\ &= \underset{\Theta}{\operatorname{argmin}}\Big(\frac{1}{2}\sum_i ||h_i(\Theta_i) - z_i||^2_{\Sigma_i}\Big)\end{aligned} \tag{3.16}$$

where the prior is drawn into the sum in the last equality. Eq. (3.16) is a *nonlinear least squares* problem which can be solved using a variety of techniques. Different approaches to SLAM apply a variety of techniques to solving this problem. The goal of iSam2 is to formulate a framework that can exploit the relationships between factor graphs, Bayes nets and Bayes trees, to modify (3.16) into a representation where marginalization and optimization is easy. Optimization is typically performed on successive linear approximations of eq. (3.16) in order to approach the minimum, Θ^*. A thorough description of how this optimization is performed along with the underlying theory of iSam2 will be presented in the subsequent chapter.

Chapter 4

ISam 2

ISam2 is an algorithm based on exploiting the relationship between factor graphs, Bayes nets and Bayes trees, and their respective matrix representations. The Bayes Tree provides a general representation of the square root information matrix, making the properties of iSam2 apply to a broader range of problems then SLAM. Regardless, it happens that the way the Bayes Tree manage to *incrementally update* a robot trajectory with a corresponding map perfectly fits the real-time demands introduced in many SLAM systems. The questions arising are why the Bayes Tree contains the information that it does and how we efficiently can construct this Bayes Tree in order to easily perform marginalization and optimization. These answers heavily rely on the theory supplied by the least-squares based approach to SLAM and the probabilistic representation of the world provided by the construction of factor graphs and Bayes nets from secs. (2.2.1) and (2.2.2).

This chapter is built up in a similar fashion as [7]. First the relationships between factor graphs and Bayes nets in a SLAM context will be presented interconnected with a description of the least-squares approach to solve the problem in eq. (3.16). Then a guide through the construction of the Bayes tree will be presented. The chapter will be concluded by a discussion on how to perform inference and marginalization in an effective manner using the presented theory.

4.1 The foundation of iSam2

At the heart of iSam2 lies the concept of least-squares. Least-squares theory provides a way to solve the MAP estimation problem formulated in eq. (3.16) by using a optimization procedure based on linearization and iterative solving. Once the general framework for how to modify the MAP estimation problem has been formulated, the improvements of iSam2 will be formulated as a solution to performing incremental marginalization and optimization.

4.1.1 Least-squares SLAM

The formulation of MAP estimation in the context of least-squares provide the foundation for the optimization problem that Isam2 wish to solve. The desired optimization problem are summarized by the nonlinear formulation presented in eq. (3.16). The approach is then to perform subsequent linearizations to obtain more manageable linear least squares problems. This is done through a Taylor expansion of the nonlinear functions $h_i(\cdot)$

$$\begin{aligned} h_i(\Theta_i) &= h_i(\Theta_i^0 + \Delta_i) \approx h_i(\Theta_i^0) + H_i\Delta_i \\ \Delta_i &:= \Theta_i - \Theta_i^0 \end{aligned} \tag{4.1}$$

where Δ_i is the state update vector and H_i is the Jacobian of $h_i(\cdot)$ in eq. (3.11) evaluated at Θ_i^0. After linearization the problem is simplified to

$$\begin{aligned} \Delta^* &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||H_i\Delta_i + h_i(\Theta_i^0) - z_i||_{\Sigma_i}^2 \\ &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||H_i\Delta_i - (z_i - h_i(\Theta_i^0))||_{\Sigma_i}^2 \end{aligned} \tag{4.2}$$

The problem can then be rewritten with the Euclidean norm rather than the Mahalanobis norm by incorporating the measurement noise in a matrix A and a vector b. To perform this conversion, the following property has to be used

$$||e||_\Sigma^2 := e^T\Sigma^{-1}e = (\Sigma^{-1/2}e)^T(\Sigma^{-1/2}e) = ||\Sigma^{-1/2}e||_2^2 \tag{4.3}$$

The linearized least squares problem, eq. (4.2), will after the conversion construct an unconstrained quadratic minimization problem. Thus it is natural to create a compound matrix A representing all the submatrices A_i and a vector b representing all vectors b_i stacked onto each other. Such a formulation is represented by the following equation

$$\begin{aligned} \Delta^* &= \underset{\Delta}{\operatorname{argmin}} \sum_i ||A_i\Delta_i - b_i||_2^2 \\ &= \underset{\Delta}{\operatorname{argmin}} ||\mathsf{A}\Delta - \mathsf{b}||_2^2 \end{aligned} \tag{4.4}$$

where A_i and b_i are represented by a scaled version of the Jacobian, H_i, and the residual, $z_i - h_i(\Theta_i^0)$, for the least squares problem.

$$\begin{aligned} A_i &= \Sigma_i^{-1/2}H_i \\ b_i &= \Sigma_i^{-1/2}(z_i - h_i(\Theta_i^0)) \end{aligned} \tag{4.5}$$

With linear formulation of the problem, eq. (4.4), it is now possible to solve this system of equations. This is typically done by applying either a Cholesky or a QR factorization to the following set of equations

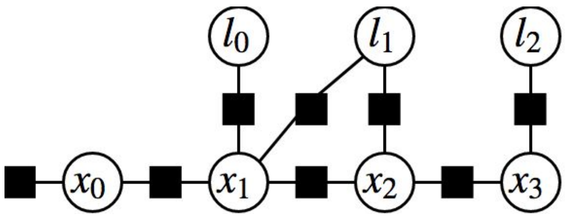


Figure 4.1: Factor graph associated with the measurement Jacobian A, with landmarks l_0, l_1 and l_2, and successive poses x_0, x_1, x_2 and x_3. Besides the factor nodes *between* successive poses and *between* landmarks and poses, there is a unary factor connected only to x_0 that express prior information.

$$(A^T A)\Delta = A^T b \tag{4.6}$$

Since $A^T A = R^T R$, by applying a Cholesky factorization, it is possible to use forward and backsubstitution to obtain a solution to the minimum of $A\Delta - b$. R is in the aforementioned expression the square root information matrix, also called the Cholesky factor.

The difference between the regular least-square SLAM approaches and Isam2 is the incremental representation of the square root information R obtained from the Bayes Tree and how it provides a way of understanding how to incrementally solve this system of equations. Thus only a part of the system is solved using the aforementioned least-squares formulation. To understand this intimate relationship between the Bayes tree and the square root information matrix, the procedure of building up the tree using properties of factor graphs, Bayes nets and their matrix representations related to the linear problem in eq.(4.4) will be described.

4.1.2 Variable elimination

It has already been established that there exists a connection between the factor graph representation and the MAP estimation problem, see sec. (3.3). With the linear expression in eq. (4.4) it turns out that the factor graph provide a way to directly understand the structure in the sparse matrix A, fig. (4.2). This matrix will be called the measurement Jacobian since each of its non-zero terms consists of a weighting of the Jacobians H_i corresponding to a measurement function, eq. (3.11). This weighting can loosely be thought of as a measure of confidence in the measurement under consideration. The reasoning behind this interpretation stems from the scaled expressions in eq. (4.5). Consequently, a similar reasoning underlie the residual terms that stack up b, which express the error between the measurement and the predicted measurement based on the current linearization point, Θ_i^0.

The measurement Jacobian presented in fig. (4.2) provide a visualization of the sparse matrix structure related to the factor graph example in fig. (4.1). As mentioned in sec. (3.3), this formulation of SLAM supports a representation of any number of variables assembled using general probability distributions or cost functions.

With the factor graph description follows a structured way of partitioning the problem. Both the matrix representation and the graphical model provide means too see how variables affect each other. To describe the inherently causal structure in SLAM and to obtain a

$$\mathbf{A} = \begin{array}{c|c|c|c|c|c|c|}
x_0 & l_0 & l_1 & x_1 & x_2 & l_2 & x_3 \\
\hline
A_{11} & & & & & & \\
\hline
A_{21} & & & A_{24} & & & \\
\hline
 & A_{32} & & A_{34} & & & \\
\hline
 & & A_{43} & & A_{45} & & \\
\hline
 & & & A_{54} & A_{55} & & \\
\hline
 & & & & A_{65} & & A_{67} \\
\hline
 & & & & & A_{76} & A_{77} \\
\hline
 & & A_{83} & A_{84} & & & \\
\hline
\end{array}$$

Figure 4.2: The measurement Jacobian corresponding to the factor graph example in fig. (4.1). *Unary function factors*, measurements of single variables, show up as lonely rows in the Jacobian. *Between function factors*, describing the relative relationship between two variables, show up as rows with two fill-ins per row.

representation more closely related to the square root information matrix, the factor graph can be modified into a chordal Bayes net. The reason for using chordal Bayes nets instead of Markov nets is to have a graphical model representing the conditional independencies in a more direct manner. The chordal Bayes net will in later sections be converted into a Bayes Tree which exploits this feature.

Thus, in terms of solving the problem at hand, building up a Bayes net from a factor graph can be considered the first step to obtain the desired representation of the square root information matrix, R. A completely general conversion from a factor graph to a Bayes net can be done using *variable elimination* of a single variable θ_j, described in alg. (1),

Algorithm 1: EliminateOne

```
function EliminateOne(($F_{j:n}$,$\theta_j$))
Remove all factors $\mathrm{f_i}(\Theta_i)$ that are adjacent to $\theta_j$
$\mathrm{S}_j \leftarrow$ all variables involved excluding $\theta_j$
$\mathrm{f_{joint}}(\theta_j, S_j) \leftarrow \prod_i \mathrm{f_i}(\Theta_i)$
p $(\theta_j|S_j)$ $\mathrm{f_{new}}(S_j) \leftarrow \mathrm{f_{joint}}(\theta_j, S_j)$
Add the new factor $\mathrm{f_{new}}(S_j)$ back into the factor graph
return p $(\theta_j|S_j)$, $\mathrm{F}_{j+1:n}$
```

where S_j is the *separator*. A separator is a set of nodes involved in the elimination step excluding θ_j that separates θ_j from the rest of the graph. Furthermore, Θ_i denotes the set defined in Section (2.2.1) rendering the index set J to include all function factors included between θ_j and S_j. Another thing to note is that the elimination is performed from what can be considered to be root nodes in the upcoming Bayes net. The reason for choosing this order will become clear after the construction of the Bayes tree.

Following the previous assumption of a Gaussian likelihood function, the product factors can now be used to eliminate one variable, θ_j, at the time. Since the goal is to calculate Δ_j in eq. (4.2) step 4 in alg. (1) is interpreted as

$$\begin{aligned} f_{\text{joint}}(\Delta_j, S_j) &= \prod_{i \in J} f_i(\Delta_i) \\ &\propto \exp\Big(-\frac{1}{2} ||\tilde{A}_i[\Delta_j; S_j] - \tilde{b}_i||_2^2 \Big) \\ &\propto \exp\Big(-\frac{1}{2} ||a\Delta_j + A_S S_j - \tilde{b}_i||_2^2 \Big) \end{aligned} \tag{4.7}$$

where $\tilde{A}_j = [a, A_s]$ is a large block-matrix accumulating all factors connected to the variable Δ_j. The new term on the RHS $\tilde{b}_j$ stacks all b_i connected to Δ_j. $\tilde{A}_j$ and $\tilde{b}_j$ are similar to A and b from eq. (4.4), but are distinguished in notation to obtain independent expressions for local measurement Jacobians and stacked residual vectors. The incremental factorization of the joint probability distribution, eq. (4.7), is equivalent to sparse QR or Cholesky factorization of the measurement Jacobian, A. This factorization results in the chain rule in step 5 of the variable elimination procedure, alg. (1)

$$\begin{aligned} & p(\Delta_j | S_j) \propto \exp\Big(-\frac{1}{2}(\Delta_j + r s_j - d)^2 \Big) \\ & r := a^\dagger A_S \\ & d := a^\dagger \tilde{b} \end{aligned} \tag{4.8}$$

where $a^\dagger$ is the pseudoinverse of a. The conditional pdf in eq. (4.8) will be the term accepted by the new Bayes net. A new factor is also obtained due to the information propagated from the eliminated variable to its neighbours. This factor represents new information to the separators from the newly eliminated variable. It is constructed as

$$\begin{aligned} & f_{\text{new}}(s_j) = \exp\Big(-\frac{1}{2} ||A' s_j - b'||^2 \Big) \\ & \theta_j = d - r s_j \\ & A' := A_s - ar \\ & b' := \tilde{b} - ad \end{aligned} \tag{4.9}$$

and will be inserted back into the factor graph. This elimination is performed from what can be considered to be the leaf nodes in the upcoming Bayes net. This allows for information to propagate from the leaf nodes and up to the root. Solving the least squares problem is the achieved by calculating the optimal assignments Δ^* in two steps. First the aforementioned functions, eqs. (4.8) and (4.9), are defined in one pass from the leaves up to the root of the Bayes net. These will then be structured in cliques during construction of the Bayes tree, representing the same causal dependencies as the Bayes net. Then, one pass down from the root, of the Bayes tree, to the leaves retrieves the optimal assignment to make up the variables Δ^*. This second pass down the tree exploits the conditional pdfs in eq. (4.8) by solving $\Delta_j = d - rs_j$ for every Δ_j using backsubstitution.

The choice of factorization in step 5 can be viewed as one of the more crucial steps in this algorithm due to the inherent property of the two methods. An interesting experiment was performed in [19], by comparing the number of non-zero elements resulting from QR and Cholesky factorization. The results show that a QR factorization produces a significantly higher amount of non-zero elements than a Cholesky factorization, thus resulting in slower computations. Albeit these staggering results, there are reasons to use a QR factorization as well, such as a low condition number. It may therefore be illuminating to see how these two methods compare in detail considering the measuremet Jacobian, fig. (4.2).

Variable elimination with QR factorization

Performing variable elimination is done by selecting a composite matrix $\tilde{A}$ and a stacked column array $\tilde{b}$ consisting of adjacent terms to the variable under elimination. For example, elimination of l_1 in (4.1) would result in the separator $s_1 = [x_1; x_2]$ and the block

$$\tilde{A}_1 = \begin{bmatrix} A_{11} & A_{13} & 0 \\ A_{21} & 0 & A_{24} \end{bmatrix} \tag{4.10}$$

The factorization described in the previous section can be re-written for QR as

$$\begin{aligned} f_{\text{joint}}(l_1, x_{1:2}) &\propto \exp\Big(-\frac{1}{2}\sum_i ||\tilde{A}_i[l_1; [x_1; x_2]] - \tilde{b}_i||_2^2\Big) \\ &= \exp\Big(-\frac{1}{2}\sum_i ||R_j l_1 + T_j[x_1; x_2] - \text{d}_j||_2^2\Big) \\ &\times \exp\Big(-\frac{1}{2}\sum_i ||A'[x_1; x_2] - b'||_2^2\Big) \end{aligned} \tag{4.11}$$

An intermediate augmented product factor matrix resulting from the factorization, eq. (4.11), will in general look like

$$[\tilde{A}_j | \tilde{b}_j] = Q \begin{bmatrix} R_j & T_j & \text{d}_j \\ & A' & b' \end{bmatrix} \tag{4.12}$$

QR can be applied directly to the measurement Jacobian, eq. (4.2), yielding $A = QR$, resulting in a solution that can be obtained through pure back-substitution without directly computing the Q matrix [7], [8], [13]. Following the procedure in the aforementioned example of variable elimination with QR, d is constructed incrementally by modifying b through d $= R^{-T}A^T$b.

Variable elimination with Cholesky factorization

Incremental Cholesky is performed by working on $A^T A$ which is in this example denoted with A_{chol}. The square root information matrix of the system at hand is constructed directly and the solution can now be obtained through forward and back substitution.

$$
\begin{aligned}
A^T A &= LL^T \\
L &= R^T
\end{aligned}
\tag{4.13}
$$

The A matrix considered in the incremental Cholesky update is a matrix consisting of affected factors. Thus, the information matrix under consideration will typically be a small relatively sparse one, with a correspondingly sparse square root information matrix provided a good variable ordering (how a good variable ordering is found is described in subsequent sections on the Bayes tree). There exists methods for directly updating the full square root information matrix using Cholesky Factorization [19], [13], but these will not be considered here.

4.1.3 The Bayes tree

An interesting property about the newly constructed Bayes net, is that it is a representation of the square root information matrix, see sec. (2.2.3). The Bayes net consist of cliques defining causal relationships between variables. These cliques present, in a graphical way, the terms directly defined by the square root information matrix (again, assuming a Gaussian measurement model). Putting together these cliques in an orderly fashion constructs the Bayes Tree [7], shown in alg. (2).

Algorithm 2: ConstructBayesTree

```
1  Function ConstructBayesTree (Bayes Net)
2      for (each conditional $p(\theta_j|S_j)$ in reverse elimination ordering) do
3          if (No parent ($S_j = \{\}$)) then
4              Start new root clique $F_r$ containing $\theta_j$
5          else
6              if ($F_{pa} \cup S_{pa}$ of parent clique $C_{pa}$ is equal to $S_j$) then
7                  Insert conditional into $C_{pa}$
8              else
9                  Start new clique $C'$ as child of $C_{pa}$ containing $\theta_j$
10             end
11         end
12     end
```

The Bayes tree is a graphical model that constructs a visual interpretation of the linear algebra applied to the square root information matrix. This graphical interface provide an easy way of performing optimization and marginalization. The Bayes tree is similar to

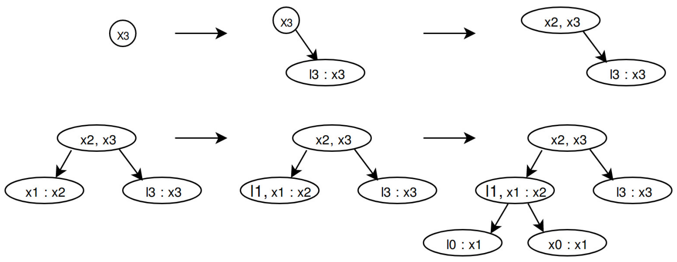


Figure 4.3: Bayes net representing causal dependencies between variable nodes.

the Junction tree in that the nodes consists of variable cliques. The difference between the trees are the directed edges of the Bayes tree. The nodes of the Bayes tree consists of two variable sets named separators, S_k, and frontal variables, F_k. Furthermore, the cliques are denoted as C_k and its parent cliques as C_{pa}. The separators are simply the nodes connecting cliques together, defined as $S_k = C_k \cup C_{\text{pa}}$, and the frontal variables are the remaining variable nodes from the Bayes net, defined as $F_k := C_k / S_k$. Exploiting this notation leads to a representation of the full joint pdf

$$f(\Theta) = \prod_k P(F_k | S_k) \tag{4.14}$$

in terms of causal dependencies between *sets of variables*. Thus each clique is represented in the graph by $C_k = F_k | S_k$, expressing the same causal dependencies as the Bayes net.

Example constructing a Bayes Tree

Consider the Bayes net in fig. (2.2). The intention is to start at the highest numbered node x_4 and construct the Bayes tree in reverse elimination ordering. The execution of alg. (2) for this example is portrayed in fig. (4.3). The frontal variables and separators in each step can be found in table (12). Note that step 6 and 7 from table (12) is concatenated in the last step in fig. (4.3).

1	$F_1 = \{x_3\}$	$S_1 = \{\}$
2	$F_2 = \{l_2\}$	$S_2 = \{x_3\}$
3	$F_3 = \{x_2\}$	$S_3 = \{x_3\}$
4	$F_4 = \{x_1\}$	$S_4 = \{x_2\}$
5	$F_5 = \{l_1\}$	$S_5 = \{x_1, x_2\}$
6	$F_6 = \{l_1\}$	$S_6 = \{x_1\}$
7	$F_7 l = \{x_0\}$	$S_7 = \{x_1\}$

The Gaussian case

As mentioned, the Gaussian case allows for a direct interpretation of the Bayes Tree in terms of the square root information matrix. How these two correlate can be viewed in fig. (4.4). Notice that the children of any clique can be ordered arbitrarily. Thus, a Bayes Tree can correspond the several square root information matrix representations. The ordering of

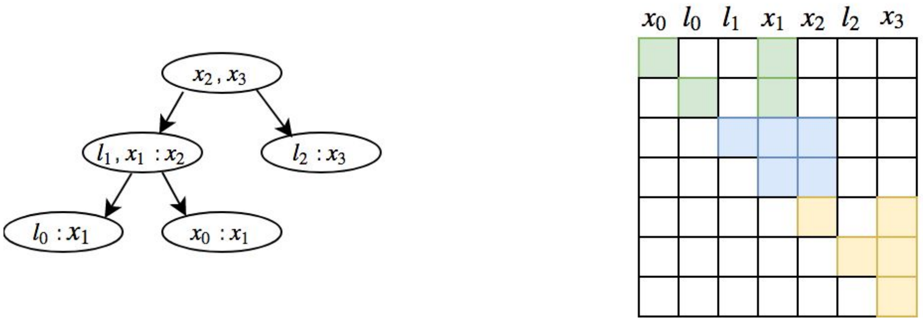


Figure 4.4: Bayes Tree and its corresponding square root information matrix with some variable ordering. The first two rows represent the variables x_0 and l_0, both with the separators x_1. The rows colored in blue are the fill-ins represented by l_1 and x_1 with separator x_2. The yellow rows represent respectively the information decoded in the root node and l_2 separated by x_3.

the children relative to the parent clique under consideration won't produce any immediate fill-in since the factorization and the numerical values will remain unchanged independent of their position in the matrix.

4.2 Innovations of iSam2

The Bayes tree makes it simpler to interpret understand that only a subset of variables in the optimization problem in eq. (4.2) has to be considered. This section is dedicated to explaining more thoroughly how the Bayes tree decodes information, how it can be used to perform inference on this subset of variables.

4.2.1 Incremental inference

A Gaussian assumption will again be applied in this section in order to provide an intuitive explanation of the incremental update step for the square-root information matrix. Updating the Bayes Tree can be done by keeping any clique considered independent and unaffected by the new variable constant during the update. Noting that dependencies in the tree propagate upwards toward the root, only the cliques between the one directly connected to the new variable and the root will be affected. The implications of this is that any other sub-trees will be rendered independent of the new variable and thereby only a subset of the square root information matrix has to be modified in order to store the change. What this information propagation says is that our new variable depends on previously added variables. For example, the current robot pose depends in some way on a pose estimated back in time. The same holds for landmarks in a SLAM context as well, since a landmark observed at a current pose will depend on the poses where that landmark was observed before.

These causal dependencies are coded into the tree because of the way the Bayes net is constructed, sec. (4.1.2), along with the reverse elimination ordering from the construction

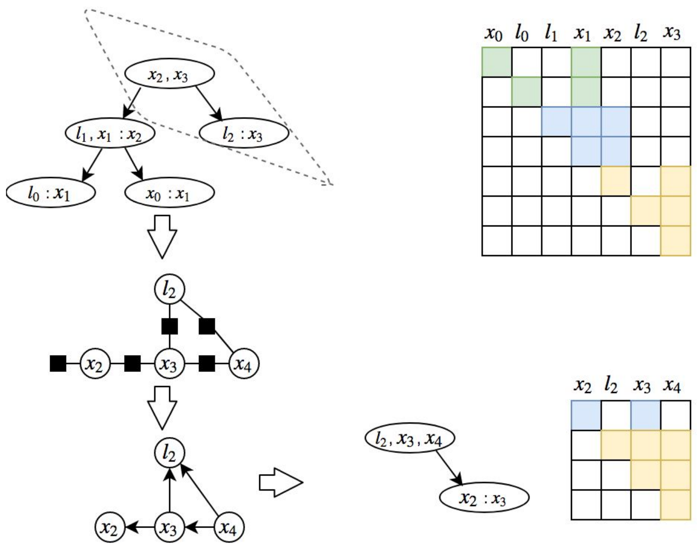


Figure 4.5: Small example of a partial state update. An affected sub-tree is decomposed into a factor graph concatenated with the new state, for then to be converted back into a new Bayes Tree. The newly constructed Bayes Tree can be defined by a local square root information factor, which can be used to compute an incremental state update.

of the Bayes tree. Since the Bayes net is constructed with arrows pointing in *reverse order* is because this propagates information from the *children* to the root node. In the example provided in sec. (4.1.2), information about all nodes will propagate up to x_3 [10]. Constructing the Bayes tree can then be performed in reverse elimination order, starting from the most recent node, since information has already been propagated. A prominent observation is that the above reasoning is also underlying the understanding of why disjoint sub-trees are conditionally independent of each other.

How a new variable is added to the graph and braided into the Bayes Tree can be seen in fig. (4.5). The matrix on the figure describes how the *partial state update* is constructed. First, affected variables at the top of tree is solved for by backsubstitution, computing a δ vector that describes how the linearization point should be updated. The square root information matrix containing the information from the local conditionals for the affected variables are shown in the bottom right corner of fig. (4.5). Secondly, it can't be assumed that the rest of the variables will stay unaffected by the new update. Thereby, it is natural to further compute state updates for descending variables until a threshold has been reached. In the small example from fig. (4.5), processing would continue one step down the left sub-tree if x_2 change by some substantial amount. This gradual expansion is allowed in the Bayes Tree due to the running intersection property. In alg. (3) the recursive procedure performing this descending update is shown.

Algorithm 3: PartialStateUpdate

```
1 Function partialStateUpdate (Bayes Tree)
2     for (Current clique $C_k = F_k : S_k$) do
3         Compute $\Delta_k$ of frontal variables $F_k$ from $p(F_k|S_k)$
4     end
5     for (All variables $\Delta_{k_j}$ in $\Delta_k$ changing by more then $\alpha$) do
6         Recursively process each descendant containing such a variable
7     end
```

4.2.2 Variable ordering

A crucial insight is that the variable ordering will affect the number of affected variables in subsequent updates. From the previous section it is known that a variable is included in the update procedure if it is included in a clique that is part of the update. Furthermore, the larger the span of variables included in the update procedure, the higher the computation requirements. Thus, it is desirable to order the variables in a way that minimizes the number of affected variables at each step. It is shown in [1] that this problem is intractable. In [7] the approach is to apply constrained COLAMD to the local subset of trees affected by the new update. Constrained COLAMD is a variable ordering approach that will locally minimize the fill-in along with exploiting an important heuristic, that newly observed variables more likely will be re-observed in the near future than old variables. It is shown in [7] that this heuristic provides a good local ordering, but suffers the same computational burden as simpler approaches in the case of large loop closures.

In fig. (4.6) a variable elimination procedure that avoids constructing the largest clique size is shown. Clique size refer to the span of the cliques, i.e, if x_6 was directly connected to x_1 in the Bayes Net, the clique size would be larger then x_4 being connected to x_1 and x_6 to x_4, as above.

4.2.3 Fluid relinearization

To avoid unnecessary relinearization of variables, it is defined a threshold β which variables will exceed if they move sufficiently far away from their previous linearization point. The reason why this functionality is implemented for the Bayes Tree is to avoid linearization and solving at each step in the update procedure. Such a procedure would require all information of the variables under consideration to be removed from the Bayes Tree, for then to be replaced by a new linearization of the nonlinear factor. Performance-wise it makes sense to avoid this extra calculation even though it comes with the cost of worse accuracy. For example, some applications may require that the updates occur with a specific rate to match real-time constraints. Contemporary tuning of β and threshold for the change in Δ, sec. (4.2.2) may therefore come in handy to match the imposed system constraints.

In addition to the complexity of relinearization itself, the re-eliminated nodes also have to contain the information passed from their sub-trees. As mentioned in sec. (4.2.1), a variable node has a causal relationship with its children, sub-trees in the Bayes Tree,

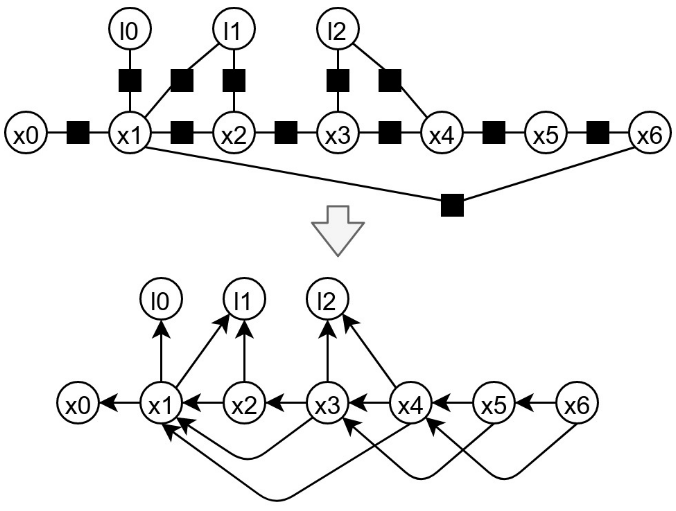


Figure 4.6: Variable elimination of an example including a small loop closure between variable nodes x_1 and x_6. A possible variable pattern that produce small clique sizes is shown above.

which the node cannot be related to if the information from the children is kept secret. In order to avoid full belief propagation where information is passed from all sub-trees to each variable relinearized variable node, a complete re-elimination of the complete system, marginal factors are cached during elimination [7]. The elimination process, variable ordering and eliminating the factor graph, can thereby be started from the middle of the tree, lowering the need information traffic through the graph.

Chapter 5

Isam2 in practice

ISam2 and the Bayes tree offer a solution to perform inference to a large variety of problems that can be modeled using Factor Graphs. Applications include general estimation problems like filtering and fixed-lag smoothing, path planning for robot manipulators, model predictive control for unmanned aerial vehicles and, the topic of this assignment, navigation and environmental mapping. Solving such tasks require knowing the limitations of the algorithm alongside the innovations described in ch. (4). Our approach in Revolve NTNU has been to implement what can be thought of as a naive or straightforward SLAM approach. The results in this chapter will show that this approach does its job, but that there is still room for improvement. Furthermore, there exist modifications of the straightforward SLAM approach which are obtained by more sophisticated treatment of the Bayes Tree data structure. This chapter will be outlined to discuss more subtle aspects of iSam2 that may improve the results presented in this chapter.

5.1 Results

The results in this section is obtained using ELD, fig. (5.1), a racing car built by Revolve NTNU. It is equipped with two cameras and a lidar which is used to provide cone positions relative to the car, fig. (1.1). An inertial navigation system (INS) equipped with a GPS is used to obtain position and velocity measurements. The test track consists of a set of cones layed out in an oval, see fig. (5.2) for illustration.

Fig. (5.3) shows the timing results from running iSam2 on the recorded data from Eld. It can be seen during loop closures that the computation time spikes up much larger then normal. These results can also be seen in the original paper on iSam2 [7], which shows the relevance of this problem. It is understood that this problem can be reduced by modifying the relinearization threshold, sec (4.2.3), thus reducing the number of variables that have to be re-computed by each loop closure, see fig. (5.4). This will however not solve the problem completely, since increasing the threshold to much may cause the solution not to converge toward ground truth, see [7].

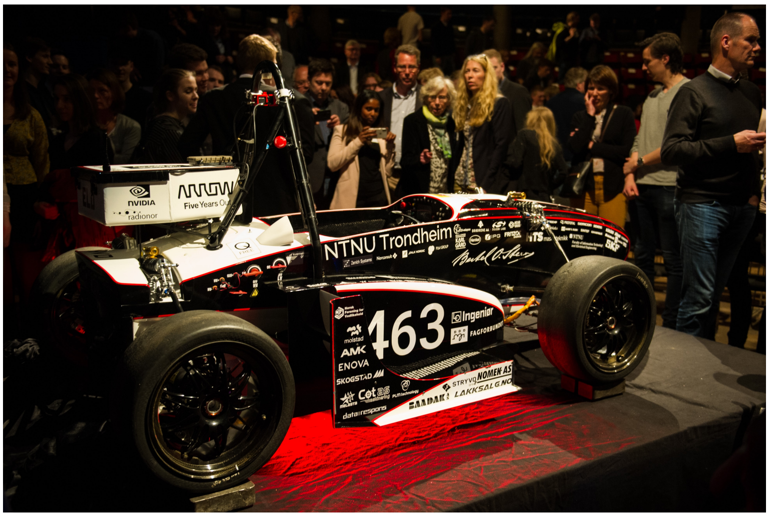


Figure 5.1: A picture of the car used for recording data for testing the iSam2.

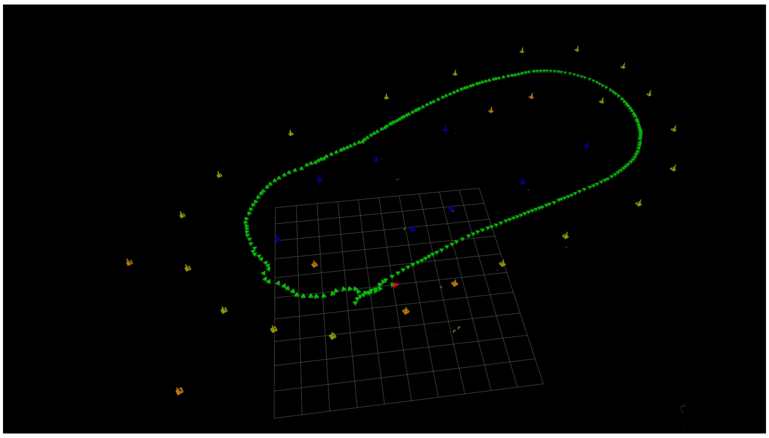


Figure 5.2: A map resulting from a driving on an oval test track with Eld. The colored circles are yellow, orange and blue cones. Note that there are a couple of outliers appearing down to the left in the map. The green arrow represents the inferred 2D pose of the car.

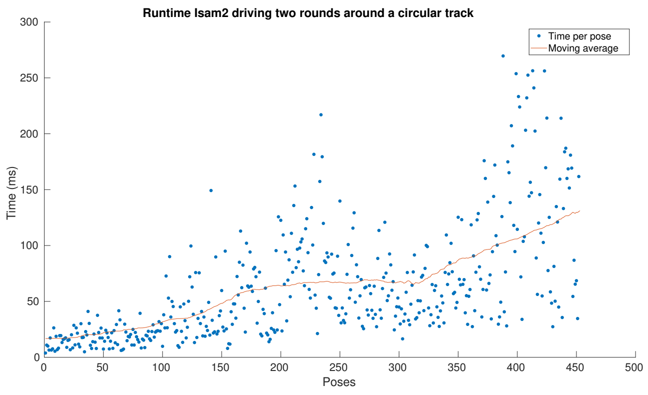


Figure 5.3: Update time for each time step using iSam2 while driving in consecutive laps with Eld. The y-axis is displays the current pose count and the x-axis is the computation time in ms. The peaks display areas where major loop closures are happening.

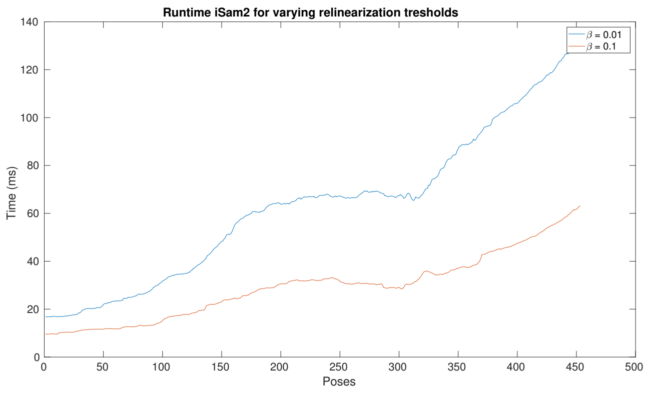


Figure 5.4: Update time for each time step using iSam2 while driving in consecutive laps with Eld with two separate relinearization thresholds.

From ch. (4) it is understood that there are a lot of parameters that can be tweaked to achieve application specific performance. These modifications can improve run-time performance considerably as seen by simply reducing the relinarization threshold. [7] presents additional clever tricks that can be done in terms of choosing the right optimization procedure and factorization method, the latter is also discussed in sec. (4.1.2). They will on the other hand not weigh out for the lacking downsides without more subtle modifications. These downsides and their possible countermeasures will be discussed next.

5.2 Concurrent filtering and smoothing

The lack of functionality provided for the case of autonomous racing can be understood by realizing that the car has to perform consecutive loop closures in a repetitive environment. The goal of SLAM in the context of autonomous racing is to map the environment with high accuracy and concurrently feed the path planner and control system with precise odometry at a high rate. The bottleneck is to provide updates at a rate consistent with the inertial navigation system (INS), which is able to provide updates at > 100 hz. To maintain such a high and reliable update rate, the system has to avoid peaks in computation which can be observed in the result section in [7]. Consequently, the peaks will increase in size as the graph increases and loop closures occur.

To avoid peaks and do partial updates at a consistently high rate, concurrent filtering and smoothing can be used [20]. This solution exploits the conditional independence between sub-trees by dividing the Bayes Tree into two from the root down, sec. (4.2.1). One sub-tree consists of a full smoother that performs loop closures and maintains a full map, whereas another sub-tree performs filtering of the most recent. Since the goal of this approach is asynchronous operation of the filter and smoother respectively, they have to synchronize at some point. How the Bayes tree is explicitly defined and how this synchronization is considered can be understood by consulting [20].

Due to time constraints, the method has not been implemented. For a more extensive explanation of the approach and to see the method compared with a state off the art batch solution, consult [20].

5.3 Sparsification

Another consequence is that of storage space of an excessive spatially dense graph. In long term operation storage space can become a problem and pruning the graph can be considered a viable option. Obviously race car driving won't affect long term operation to the extent that storage space becomes a problem, but it is pragmatically desirable due to the upsides of reducing the number of variables in the system. Reducing the number of variables will in turn imply a Bayes Tree consisting of fewer cliques, thus shorter sub-trees, which results in smaller clique sizes. It will consequently affect the belief propagation performed during variable elimination, sec. (4.1.2), the variable ordering, sec. (4.2.2), by reducing dimensionality of the problem.

The question then is whether it is possible to optimize the graph to require a *minimal* amount of storage without losing a significant amount of information. In the SLAM literature [3], node and edge sparsification is well represented. To the authors knowledge, this functionality isn't implemented in the current version of GTSAM applied to iSam2. There exists an approach for performing *consistent sparsification*, but this is only for the batch related solution iSam, see [6].

From the authors viewpoint, pruning is possible within the Bayes Tree framework by exploiting the fact that leaf nodes are easily marginalized. This is understood with by the fact that leaf nodes have passed all their information upwards toward the root, during construction of the Bayes tree, see sec. (4.2.1). Therefore, they can be removed without affecting the system. The downside appear when it is desirable to prune away a node that is in the middle of the tree. This node would have to be shuffled down the tree by changing the variable ordering, and reconstructing the Bayes tree with this new ordering. This shuffling of nodes may result in detrimental effects in terms of increased fill-in, see Figure 2 in [20], and thus a large increase in computation [7]. Dependent on the application this may be undesirable due to the increased amount of computational spikes. On the other hand, considering a combination of such a sparsification within the smoother presented in, sec. (5.2), may provide a solution to both long term and fast real-time operation.

Chapter 6

Conclusion

In this assignment SLAM and iSam2 has been presented with the intention of making it digestible for someone with a purely statistical background. First some graphical models were presented with the intention of presenting the required prerequisites to understand the MAP estimation approach to SLAM as well as the attributes of the iSam2 algorithm. Then SLAM was presented first an informal fashion before its underlying statistical structure was presented more rigorously. With the fundamental knowledge of SLAM out in the open, iSam2 was presented with a goal of presenting the method used by Revolve NTNU on their driverless car.

It can be concluded that the straightforward SLAM approach currently in use have issues with computational spikes along with linearly increasing memory consumption, sec. (5.1). The computational spikes can result in disastrous effects within high speed applications, like collisions, if not handled properly. The linearly increasing memory may not be a problem in the application of race car driving, but for a more long lived robot it may be desirable to do something about. Further work includes an implementation of concurrent filtering and smoothing [20] to guarantee more reliable operation along with researching how sparsification may be implemented effectively for iSam2 within the GTSAM framework. A solution combining sparsification with concurrent filtering and smoothing will also be looked into.

Bibliography

[1] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (SLAM): part II. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, September 2006.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *Robotics, IEEE Transactions on*, 32(6):1309–1332, December 2016.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *Robotics Automation Magazine, IEEE*, 13(2):99–110, June 2006.

[5] J. E. Guivant and E. M. Nebot. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Transactions on Robotics and Automation*, 17(3):242–257, Jun 2001.

[6] G. Huang, M. Kaess, and J. J. Leonard. Consistent sparsification for graph optimization. In *2013 European Conference on Mobile Robots*, pages 150–157, Sept 2013.

[7] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, February 2012.

[8] M. Kaess, A. Ranganathan, and F. Dellaert. isam: Incremental smoothing and mapping. *Robotics, IEEE Transactions on*, 24(6):1365–1378, December 2008.

[9] J.-H. Kim, S. Sukkarieh, and S. Wishart. *Real-Time Navigation, Guidance, and Control of a UAV Using Low-Cost Sensors*, pages 299–309. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[10] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, Feb 2001.

[11] M. Montemerlo and S. Thrun. *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics*, volume 27 of *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[12] J. Pearl. Probabilistic reasoning in intelligent systems : networks of plausible inference, 1988.

[13] L. Polok, M. Solony, V. Ila, P. Smrz, and P. Zemcik. Incremental cholesky factorization for least squares problems in robotics. *IFAC Proceedings Volumes*, 46(10):172–178, June 2013.

[14] H. Rue. Gaussian markov random fields : theory and applications, 2005.

[15] A. Stevens, M. Stevens, and H. Durrant-Whyte. ldquo;oxnav rdquo;: reliable autonomous navigation. In *Proceedings of 1995 IEEE International Conference on Robotics and Automation*, volume 3, pages 2607–2612 vol.3, May 1995.

[16] S. Thrun. Probabilistic robotics. *Commun. ACM*, 45(3):52–57, Mar. 2002.

[17] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, 23(7-8):693–716, 2004.

[18] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. Van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23(9):661–692, September 2006.

[19] X. Wang, R. Marcotte, G. Ferrer, and E. Olson. Aprilsam: Real-time smoothing and mapping.

[20] S. Williams, V. Indelman, M. Kaess, R. Roberts, J. J. Leonard, and F. Dellaert. Concurrent filtering and smoothing: A parallel architecture for real-time navigation and full smoothing. *The International Journal of Robotics Research*, 33(12):1544–1568, October 2014.

[21] J. Zhang and S. Singh. Low-drift and real-time lidar odometry and mapping. *Autonomous Robots*, 41(2):401–416, February 2017.